**UNIVERSIDAD DE ALCALÁ**

**Escuela Técnica Superior de Ingeniería Informática**

**INGENIERÍA INFORMÁTICA**

**Trabajo Fin de Máster**

# Documento de pruebas interactivo en la metodología ágil Scrum

**Autor: Diego Alexander Esquivel Romero**

**Profesor Tutor: José Carlos Ciria Cosculluela**

**Septiembre 2020**

**UNIVERSIDAD DE ALCALÁ**

**Escuela Técnica Superior de Ingeniería Informática**

**INGENIERÍA INFORMÁTICA**

**Trabajo Fin de Máster**

**Documento de pruebas interactivo en la metodología ágil Scrum**

**Autor: Diego Alexander Esquivel Romero**

**Profesor Tutor: José Carlos Ciria Cosculluela**

**Firmas**

**Secretario de la**

**CALIFICACIÓN: .....**

**FECHA: .....**

**Carta de aceptación**

## Abstract

**University of Alcalá**

**Department of Computer Science**

Agile Scrum is one of the most popular frameworks for software development implemented worldwide. It holds a variety of roles that interact to achieve successful projects such as the **Scrum Master**, the **Product Owner** or the **Development Team** (which will be explained in detail in the theoretical framework). Each role has specific responsibilities within the development of a project.

In order to perform the technical actions within the **Development Team** there are essential profiles that should cover the aforementioned tasks: Developers specialized in different technologies, Database Managers or Test Managers. They all are part of a cross-functional team in which the knowledge is shared and functions can be assumed by any of the team members.

The aim of this TFM is to give a complete approach of the need of a specific role responsible for the tests, which should exist in the project at all times though it might break the Scrum recommendation that there should not exist fixed positions within the team. Also, a new responsibility of the aforementioned **test manager** will be specified, consisting in the delivery of documented integration testing, including verified cases or pending endorsement. Thus, the project status will be accessible at all times. This document will function as a log to facilitate test iteration, if necessary, and future maintenance stages.

## Resumen

El framework ágil Scrum es el más implantado a nivel mundial para el desarrollo de software. Para este framework existen diversos roles que interactúan para lograr éxito en los proyectos desde el **Scrum Máster**, el **Product Owner** y el **Development Team** (los cuales se explicaran con detenimiento en el marco teórico) y cada uno de ellos posee responsabilidades específicas dentro del desarrollo de un proyecto.

Para realizar las acciones técnicas dentro del **Development Team** hay perfiles necesarios que deben de cubrir las acciones antes mencionadas, desde desarrolladores en diversas tecnologías, pasando por gestores de bases de datos hasta responsables de pruebas buscando siempre generar un equipo multi-funcional en el que el conocimiento sea compartido y las tareas puedan ser asumidos por cualquiera de los miembros del equipo.

En este TFM se plantea la necesidad de tener un rol especifícame **responsable de las pruebas** el cual debe existir en todo momento en el proyecto, rompiendo un poco con la recomendación Scrum de no existir roles específicos.

Y adicionar una nueva responsabilidad al mencionado responsable de pruebas, la cual debe ser entregar un documento específico de pruebas de integración con los casos verificados o pendientes de verificar el cual debe reflejar el estado del proyecto en todo momento. Este documento funcionará como bitácora para facilitar repetir las pruebas en caso de ser necesario, pero también será entregado para etapas de mantenimiento posteriores.

*“El éxito es aprender a ir de fracaso en fracaso sin desesperarse”*

Winston Churchill

# Agradecimientos

*A mi hija y a mi esposa.*
*Por su apoyo incondicional, por estar siempre a mi lado dándome sonrisas y compañía.*

*A mis padres y a mi hermana.*
*Por nunca desfallecer, por siempre darme apoyo y consejos.*

*A mis amigas Elena y Angela.*
*Por su apoyo, consejos y tiempo para hacer esto posible.*

*A mi tutor*
*Gracias por darme ánimo y orientaciones desde el principio y por no permitir que dejara de intentarlo.*

# Contenidos

## Índice de Figuras



E.P. = Elaboración propia

[n]+E.P. = Elaboración propia a partir de la referencia [n]

# Hipótesis

¿Puede la falta de calidad ser un problema en los desarrollos guiados por las frameworks o metodologías ágiles? ¿Cómo es el mantenimiento de los proyectos desarrollados bajo esos frameworks o metodologías a medio y largo plazo? ¿Evoluciona fácilmente este tipo de proyectos después de su entrega? Estas y otras muchas preguntas aparecen cuando se plantea la adopción de metodologías ágiles por parte de las compañías, sobre todo cuando se viene de modelos desarrollos en esquemas tradicionales.

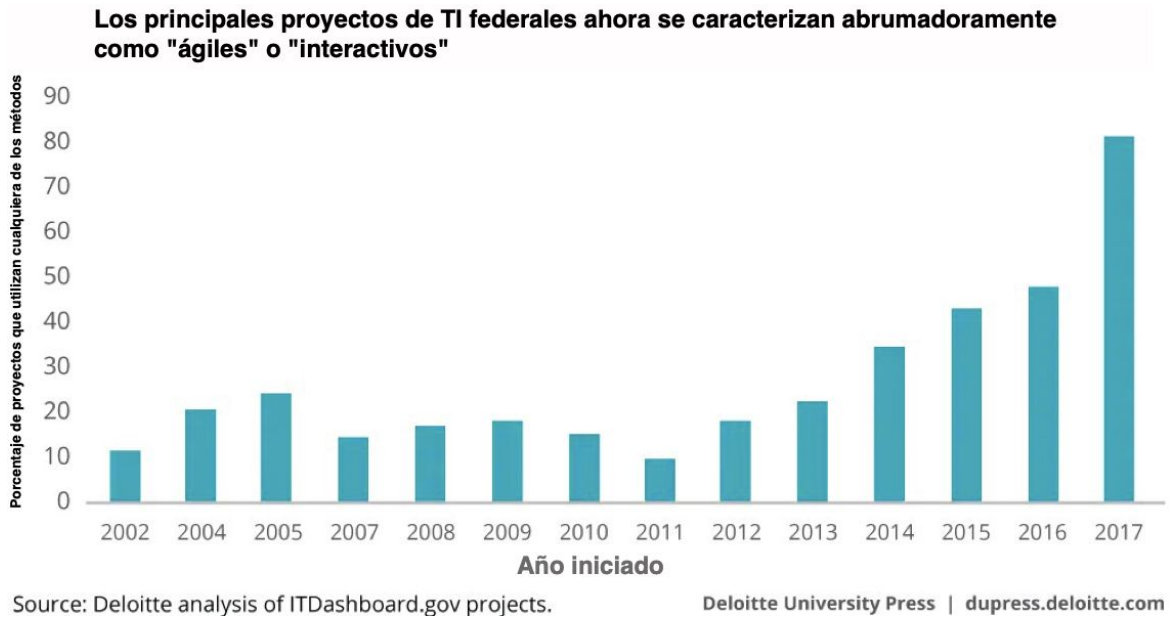


*Figura 1. Agilidad en cifras Un análisis de datos del desarrollo ágil en el gobierno federal de EE. UU., 2018 [1]*

En la actualidad las metodologías ágiles de desarrollo de software experimentan un boom de grandes dimensiones. Con banderas tan llamativas como la flexibilidad, deslocalización, adaptabilidad al cambio, la velocidad, la entrega de valor en instancias muy iniciales del desarrollo y por qué no decirlo, el ahorro significativo de tiempo en tareas poco valoradas. Son beneficios extremadamente llamativos para la

gran mayoría de empresas y organismos gubernamentales que desean crear y actualizar sus desarrollos software.

La necesidad de estas metodologías no está en discusión, máxime con la situación particular del mundo actualmente.[1]

Sin embargo, experiencias acontecidas en diversos proyectos siempre han manifestado dudas por diversas casuísticas que parecen no estar lo suficientemente maduras o se prestan a ambigüedad en las metodologías ágiles, particularmente SCRUM[2] (tener en cuenta que actualmente es la metodología ágil más adoptada por las diversas instituciones ver en la figura 2). Y esas dudas están en la inexistente o insuficiente documentación de pruebas las cuales deben soportar la calidad del producto y que salvo requerimiento específico del proyecto la misma no es desarrollada.

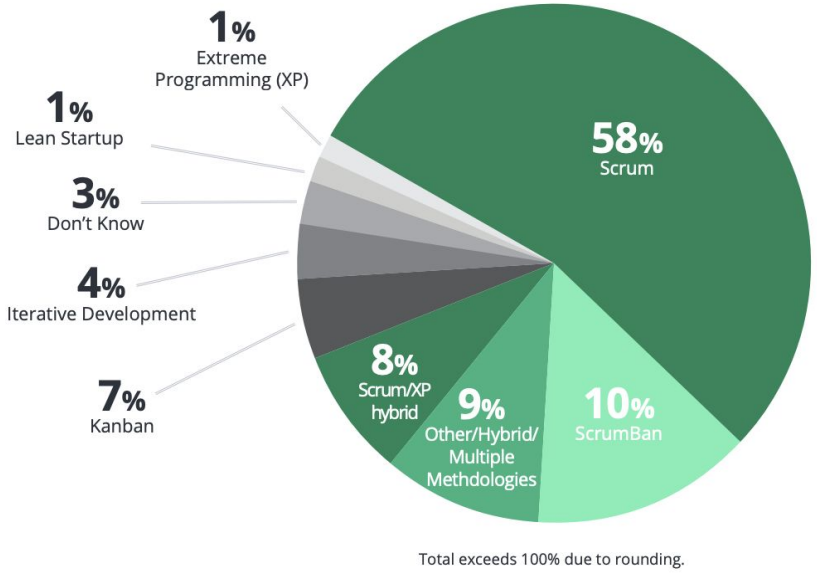


*Figura 2. Estadísticas de adopción de las metodologías y frameworks ágiles [2]*

[1] En el momento del desarrollo de este TFM se estaba viviendo la pandemia de COVID-19 la cual como uno de sus efectos deslocalizo el trabajo.

[2] Scrum es un marco de trabajo iterativo e incremental para el desarrollo de proyectos, productos y aplicaciones. Estructura el desarrollo en ciclos de trabajo llamados Sprints. Son iteraciones de 1 a 4 semanas, y se van sucediendo una detrás de otra. Los Sprints son de duración fija – terminan en una fecha específica, aunque no se haya terminado el trabajo, y nunca se alargan. Se limitan en tiempo.

Los problemas encontrados debido a esto no son pocos y no discrimina el tamaño del producto, complejidad, tipo de negocio y menos aún el estado del desarrollo del producto; debido a que los problemas que esta falta de documentación genera, se identifican tanto en desarrollos de producto a penas iniciado, como en etapas muy avanzadas de mencionado desarrollo o incluso en proyectos con largo tiempo siendo productivos.

El planteamiento que pretendo defender en este TFM es como tareas claves de documentación de pruebas vistas en metodologías consideradas no ágiles o tradicionales como son CMMI[3] pueden ser usadas como guías de buenas prácticas para mejorar significativamente la trazabilidad y la calidad de los productos desarrollados en Scrum y sobre todo su posterior mantenimiento y evolución.

[3] CMMI (Capability Maturity Model Integration) es el modelo de CMMI que brinda un conjunto de guías completas e integradas para desarrollar productos y servicios, además de brindar orientación para aplicar las buenas prácticas en una organización de desarrollo.

## Ámbito

El ámbito de este TFM se centra en los proyectos desarrollados con el framework ágil Scrum cuya arquitectura está orientada al desarrollo y/o mantenimiento de microservicios y cómo se facilitaría considerablemente la verificación del producto en cualquier etapa del desarrollo, además del mantenimiento posterior a la entrega.

Dentro del mismo, se realizarán sugerencias y recomendaciones aplicables tanto a productos que se encuentren en etapas productivas como en etapas iniciales de desarrollo.

## Justificación

La calidad en el software o en cualquier otro producto es el grado en el que un conjunto de características inherentes de un producto cumple con los requisitos con los que fue creado. Si además de querer calidad, se aspira a que esa calidad se conserve de la mejor manera a lo largo del tiempo, esos son en mi concepto, los grandes hitos o puntos de mejorables en algunas metodologías ágiles. El desarrollo de este TFM estará centrado en desarrollar algunas ideas de mejora sobre el framework ágil Scrum, el cual no solo es el framework más implantado a nivel mundial, sino que es en el que tengo más experiencia profesional.

A pesar de ser una metodología que ha elevado el desarrollo de productos software a nuevos niveles, es importante tener en cuenta que el manifiesto ágil[4] (raíz neural y guía de todas las metodologías ágiles[5]) fue publicado en 2001 (19 años a la fecha) y el desarrollo de productos software se lleva realizando desde mediados del siglo XX (más de 70 años). La comparación de edades la realizó porque me surgió la siguiente pregunta ¿pueden existir en las metodologías "tradicionales" ideas o conceptos que nos puedan aportar valor para mejorar la calidad de los productos ágiles?

El modelo tradicional de desarrollo de software en cascada nació como una adaptación del modelo de creación de proyectos de amplia envergadura tangibles tales como la creación de coches, aviones, edificios en la creación de productos software. En los modelos tradicionales las fases de análisis, diseño, implementación, pruebas y documentación eran de vital importancia y alrededor de ellos nacieron modelos relacionados con la mejora continua del software tales como

[4] El manifiesto ágil es la semilla que permitió el desarrollo de todas las metodologías ágiles que tenemos en la actualidad, es un documento con un lineamiento claro de ideas que sirven como guía y base para el concepto de agilidad en el desarrollo de software.

[5] Por metodologías ágiles se entiende que son aquellas que permiten adaptar la forma de trabajo a las condiciones del proyecto, buscando flexibilidad y rapidez en la respuesta para amoldar el proyecto y su desarrollo a las circunstancias específicas del entorno.

CMMI, EFQM entre otros. Estos modelos hacían de la calidad, trazabilidad y la documentación hitos que definían un proyecto como exitoso.

En los casos que se mencionan a continuación, pretendo abordar las siguientes preguntas:

- ¿Es requerido un rol específico responsable de las pruebas y todo lo relacionado con las mismas? ¿cómo se incorporaría en el modelo ágil de Scrum?
- ¿Se debería crear documentación clave de pruebas en el framework de Scrum que ayude a facilitar la evolución del producto a largo plazo?
- ¿Cómo se recomienda la incorporación del desarrollo del documento de pruebas en las etapas de Scrum?

# Marco Teórico

## 1. Qué es un microservicio y que es una aplicación monolítica

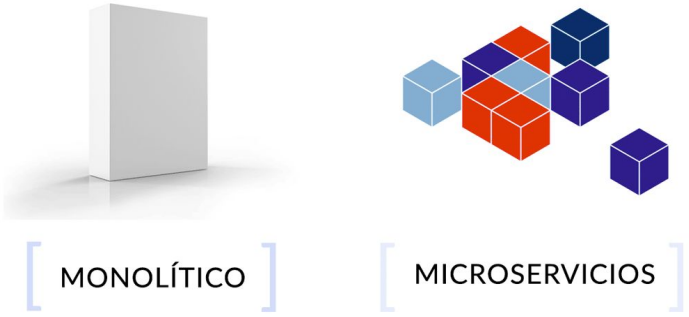


*Figura 3. Aplicación monolítica vs aplicación orientada a microservicios [3]*

Una **aplicación monolítica** es una aplicación en la que todos los componentes requeridos para su funcionamiento están integrados en un único paquete de despliegue como se puede ver en la figura 4, desde las interfases de usuario, los accesos a bases de datos, accesos externos, lógica de negocio, etc. Las aplicaciones monolíticas son completamente independientes lo que las hace autónomas y autosuficientes.

Por el contrario, los **microservicios** implican unificar funcionalidad y lógica en aplicaciones más pequeñas y totalmente autosuficientes con sus propias interfases API, lógica de negocio y acceso a base de datos como se puede apreciar en la figura 4. Lo que les permite ser fácilmente escalable en caso de necesitar mejoras, consumibles por otros microservicios o aplicaciones y reparables de manera independiente en caso de identificar errores.

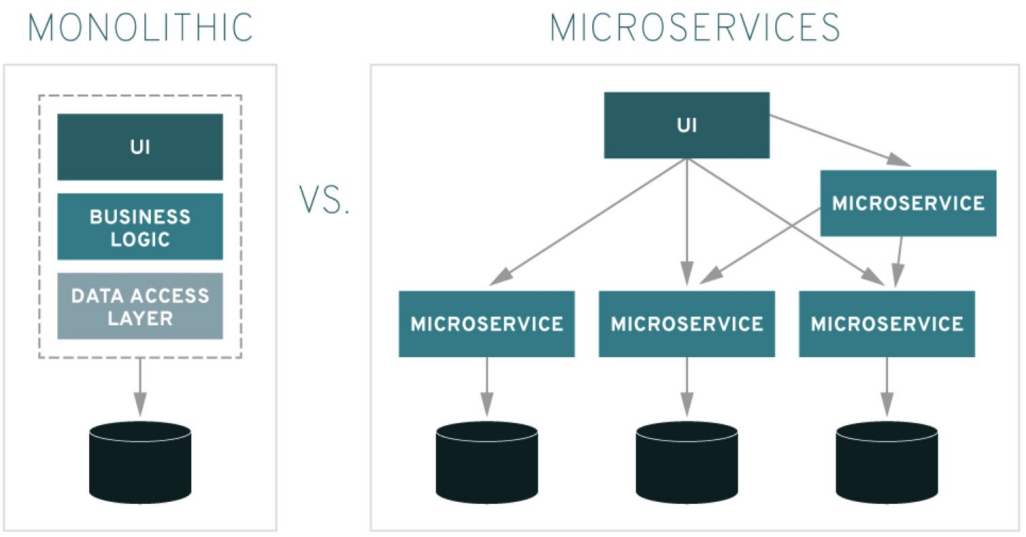


*Figura 4. Esquema comparativo entre aplicación monolítica y aplicación basada en microservicios [8]*

El enfoque tradicional de **arquitectura monolítica** en el software ha sido utilizado para el desarrollo de aplicaciones web desde sus inicios. La construcción de estos no está acotado a un único equipo de desarrollo, pero si responde a un diseño centralizado de rápido crecimiento y expuesto a cambios permanentes. A medida que crecen las aplicaciones monolíticas, son susceptibles cambios que pueden impactar la complejidad y dificultan el mantenimiento, aumentando no solo los tiempos en las diversas fases (análisis, diseño, implementación y pruebas), sino también los costos en el caso de ser necesario un escalamiento.[6]

En la actualidad la **arquitectura monolítica** está siendo sustituida poco a poco por arquitecturas basadas en microservicios, la cual es una tendencia emergente que surge de las necesidades de la industria de software, el cual ofrece las siguientes ventajas:

1. Reducción de ciclos de desarrollo, lo que mejora significativamente los tiempos de entrega de valor.

[6] IEEE - “Methodology to transform a monolithic software into a microservice architecture”

2. Escalabilidad: cuando crece la demanda, la infraestructura puede ajustarse fácilmente.
3. Facilidad de recuperación: Si los microservicios son correctamente independientes la caída de uno de ellos no afecta al resto.
4. Facilidad de implementación: Al ser aplicaciones modulares, sus desarrollos son más pequeños y controlados, lo que implica que se debe tener más coordinación, pero menos complejidad propia.
5. Accesibilidad: Debido a su modularidad, las revisiones por piezas son más fáciles que en aplicaciones tradicionales.
6. Aplicaciones abiertas: Gracias al uso de API`s[7], los microservicios no requieren un lenguaje de desarrollo único y permiten el uso de diversas tecnologías.

Motivos por los que ha tenido una aceptación importante principalmente en empresas líderes de los diversos sectores tecnológicos, como lo son: Banca, Seguros, turismo además de titanes de la tecnología como lo son Google, Amazon, Netflix etc.

[7] Es la abreviatura de Application Programming Interfaces, que en español significa interfaz de programación de aplicaciones, es una especificación formal que establece cómo un módulo de un software se comunica o interactúa con otro para cumplir una o muchas funciones.

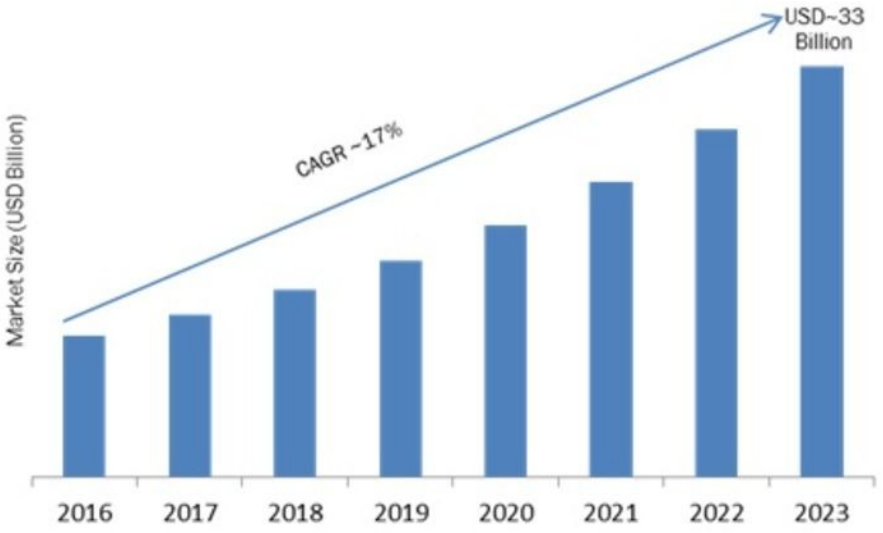


*Figura 5. Estimado de crecimiento de inversión del mercado en la arquitectura de microservicios [4]*

La arquitectura de los microservicios se valora sobre todo el nivel de detalle, la simplicidad de comunicación, la sencillez de infraestructura y la capacidad para aportar valor entre aplicaciones.

El objetivo final de los microservicios es ser fácil de consumir por otras aplicaciones o microservicios además de simplificar la evolución individual sin impactar los requerimientos que ya se tienen implementados. El desarrollo de los microservicios ha permitido que cada uno de ellos no solo sea un proveedor, sino que gracias a su estructura puede ser él mismo un consumidor de otro microservicio o entidad externa.

## 2. Scrum

El framework Scrum describe una serie de prácticas para el desarrollo de proyectos software que alteran radicalmente los modelos tradicionales de gestión de proyectos, esto se logra minimizando la burocracia e incrementando la integración en los involucrados mejorando de esta manera la rapidez con la que se adapta a los cambios.

En la siguiente figura 6 obtenido de la página web Scrum.org se puede observar los roles, componentes y eventos del framework, los más relevantes para tener el contexto se explicarán a alto nivel a continuación.

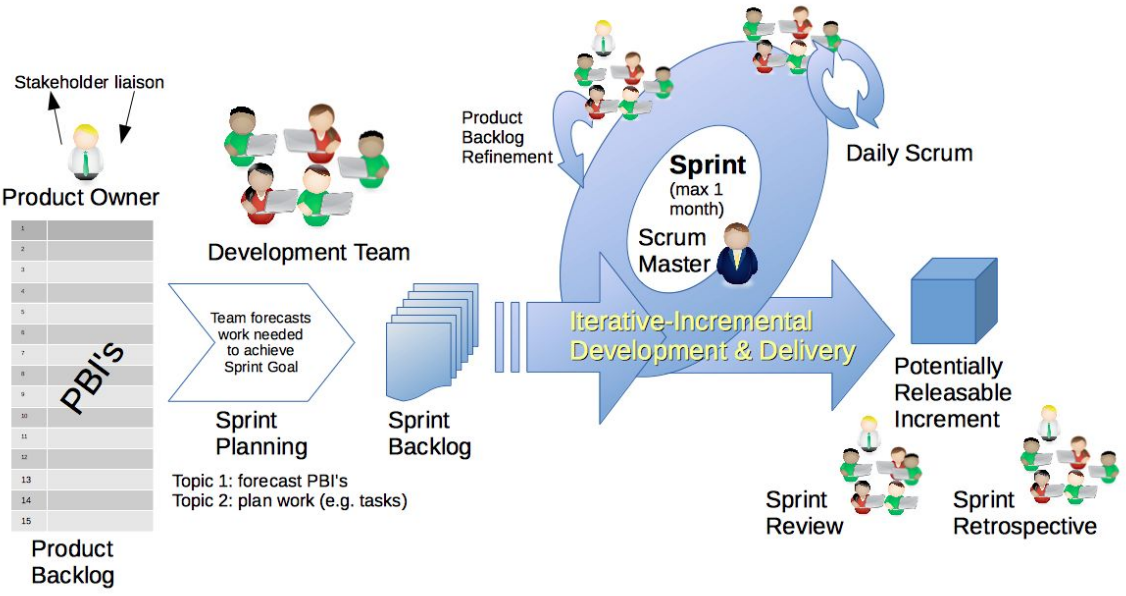


*Figura 6. Sprints y como encajan en el modelo Scrum [5]*

**Scrum** divide al proyecto en pequeñas interacciones de entre 1 y 4 semanas llamadas sprints y la idea es gestionar los mencionados **sprints** con el fin de obtener incrementos significativos y realizar adaptaciones rápidas en caso de cambios inesperados.

**Potentially Releasable Increment**

Se llama así a los resultados funcionales de cada sprint y el valor que aportan esos resultados del proyecto.

**Product Backlog**

Es un listado en orden de prioridad de todos los requerimientos (tanto funcionales como no funcionales) y resultados esperados o PBI´s (Product Backlog Item) que harían que el proyecto fuera exitoso, los anteriores creados y obtenidos mediante la planificación llevada a cabo por el Product Owner (Los roles se explicaran posteriormente) con todos los Stakeholders. Cada PBI debería contener el mayor nivel de detalle del requerimiento de negocio.

**Sprint Backlog**

Son las tareas que se realizarán en los sprint, luego que el equipo decida y sean consensuadas las mismas con el product Owner. Las tareas deben estar en orden de prioridad para ser trabajadas. Las tareas del *sprint backlog* son las tareas al detalle que se requieren para realizar las tareas del *product backlog*, de manera similar a como se ven en la figura 6.

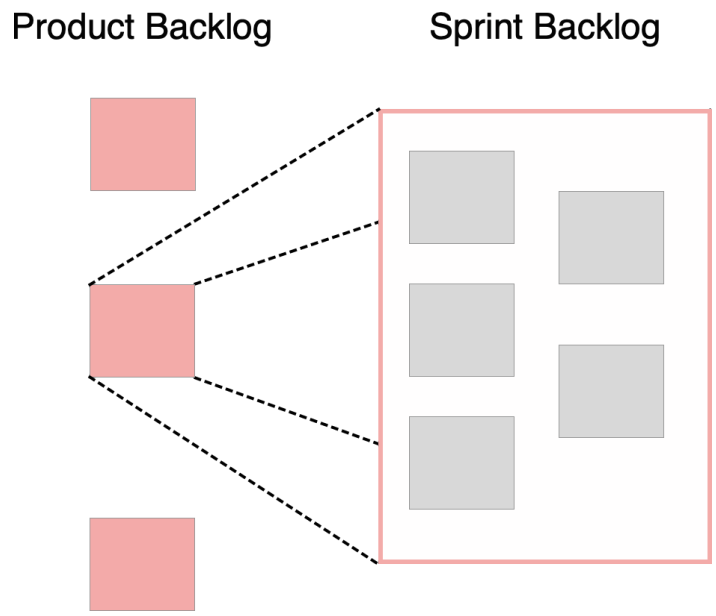


*Figura 7. Product backlog vs sprint backlog*

**Definition of Done**

Hace parte del sprint y lista los criterios que permiten evaluar si el trabajo realizado provee un incremento en el valor del producto que se está desarrollando. Los criterios deben estar claros para el equipo de desarrollo y deben estar alineados con la planificación del desarrollo del product Owner.

**Sprint Planning**

Es una reunión de no más de 8 horas donde se deben seleccionar las tareas del product Backlog que van a ser trabajadas en el sprint. Las tareas seleccionadas deben ser adicionadas al sprint backlog, en esta reunión deben explicar cada una de las tareas del sprint de manera clara y concisa para el Development Team.

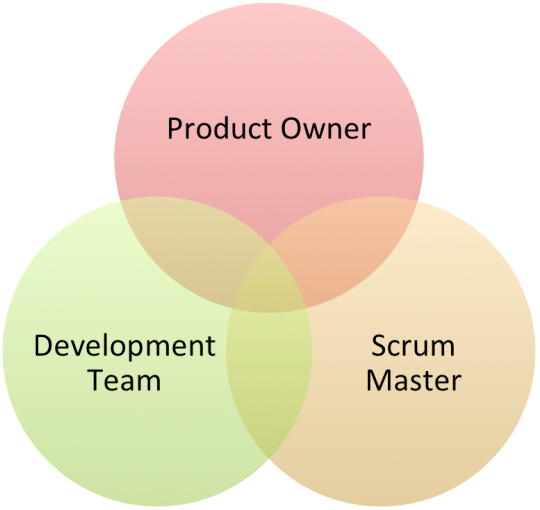


*Figura 8. Roles en el equipo Scrum [6]*

El Scrum Master es un nuevo tipo de gestor. Es la persona responsable de que todas las prácticas del framework se pongan en funcionamiento y de ser necesario realice las adaptaciones necesarias para que las mismas no interfieran en el correcto flujo de trabajo del Development Team. El Scrum Máster no tiene autoridad real, pero sí actúa como un facilitador del trabajo evitando que los impedimentos bloqueen la consecución de objetivos.

El Product Owner es quien debe mantener actualizado el Product Backlog y representa al Stakeholder delante del Development Team. Es él quien define las características del producto, pero no tiene autoridad de líder de equipo ni de injerirse en cambios en el sprint una vez iniciado.

El Development Team se conforma usualmente por entre 3 y 9 miembros quienes deben realizar todas las tareas necesarias para completar el producto sprint a sprint, los miembros del equipo deben de tener las habilidades profesionales necesarias para completar las tareas pruebas, desarrollo, bases de datos, etc. Es importante tener en cuenta que para scrum el development team debe ser multi-funcional (cualquier miembro del equipo debe poder acometer cualquier tarea).

Los Stakeholder representan a los clientes en Scrum, solo deben tener interacción con el Product Owner y será el quien represente sus intereses en el proyecto. Pueden ser desde patrocinadores del proyecto como clientes finales o consumidores del producto.

## 3. Modelo tradicional de desarrollo CMMI

El modelo CMMI (Modelo de Madurez de Capacidades de Integración) permite medir o establecer qué tan madura es una organización o equipo de trabajo usando como base el mismo modelo y como sus recomendaciones son implantadas.

Dentro del mismo se presenta una guía de características efectivas integrables en los procesos de las organizaciones proporcionando de esta manera una guía o camino de ruta para la creación o mejora de los procesos internos. El objetivo final es facilitar a una organización el desarrollo de productos o soluciones mejorando su capacidad para gestionar el desarrollo, costes, adquisición y el mantenimiento de sus productos o servicios.

CMMI es un modelo que dice **qué hacer**, pero **no cómo hacerlo**. No es un conjunto de prácticas y metodologías, son recomendaciones y tareas que influyen los procesos y que le permitirán a la organización madurar y mejorar en todas sus áreas relacionadas con el desarrollo de nuevos productos.

El motivo por el cual en ocasiones presenta complejidad a la hora de implantarlo puede ser debido a que todas las herramientas incluidas en el proceso de desarrollo deben tener lugar dentro del modelo o de sus objetivos y de una u otra manera deben ayudar a cumplir los “QUÉ” de cada nivel de madurez.

### ¿Qué beneficios nos brinda implantar el modelo?

Los beneficios que se buscan al implantar correctamente CMMI son los siguientes:

- Desarrollar el producto dentro de tiempo y costo.
- Mejorar la productividad de los procesos de desarrollo y la satisfacción del cliente.

- Mejora el tiempo de entrega a producción de nuevos productos o mejoras posteriores a la entrega.
- Minimiza el coste de mantenimiento por productos heredados.

**¿Dónde puede ser usado el modelo CMMI?**

CMMI puede ser usado en cualquier organización de cualquier ámbito y tamaño desde gobierno, hasta aviación, sanidad, banca etc, siempre y cuando la misma desee mejorar las capacidades y el rendimiento. Muchas organizaciones usan los procesos creados u optimizados para CMMI con el fin de desarrollar, mejorar, mantener y/o adquirir productos y servicios.

CMMI se representa con **niveles de madurez** y los mismos se logran en la medida en que se cumplan las metas específicas y genéricas asociadas a cada área de procesos.

**Niveles de madurez**

Son la ruta que garantiza que las organizaciones puedan mejorar sus áreas de proceso progresivamente para hacerlas cada vez más exitosas. Cada nivel de madurez puede ser certificado por CMMI e incorpora un conjunto de procesos que si se implementan ayudarán a alcanzar un nivel de madurez completo. Como se comentó anteriormente es un camino progresivo por lo que no se puede alcanzar un nivel si antes tener el nivel previo, debido a esto no se puede llegar al nivel 3 sin tener todas las practicas del nivel 2 y así sucesivamente.


*Figura 9. Niveles de madurez CMMI*

**Nivel 1. No gestionado:** todos los procesos son caóticos o reactivos, la organización no proporciona un entorno estable para dar soporte a los procesos, los logros se obtienen por la habilidad individual de las personas.

**Nivel 2. Gestionado:** siguen existiendo procesos caóticos, aunque ya existe gestión en los proyectos, los mismos se planifican y se ejecutan desacuerdo con políticas creadas dentro de la organización para este fin y se dispone de personal cualificado para producir resultados controlados.

En el mismo se realizan tareas de monitorización y control lo que permite ser reactivos y tomar decisiones para reencausar desviaciones de tiempo y coste. En este nivel se pueden alcanzar compromisos entre las partes interesadas y los mismos son ajustados según lo requiera las exigencias del proyecto.

**Nivel 3. Definido:** Cuando se llega a este nivel los procesos están bien comprendidos y documentados, se siguen estándares, procedimientos, métodos y se han implantado prácticas obligatorias de ingeniería que facilitan la gestiona el proyecto de manera eficiente.

Este es el nivel de madurez en el que se pueden definir herramientas automáticas de verificación de código, automatización de pruebas con las que se realizan verificaciones tanto del proceso de construcción como de requerimientos del producto.

**Nivel 4. Administrado cualitativamente (o medido):** Los proyectos junto con la organización están orientados de manera cuantitativa hacia la calidad y el rendimiento del proceso. Se recopilan métricas que permiten medir los resultados tanto del proyecto como de los procesos que han estado involucrados en el desarrollo del mismo. La calidad y el rendimiento se mide estadísticamente de la misma manera que retroalimenta todos los procesos para su análisis.

**Nivel 5. Optimizado:** Revisan y mejoran continuamente los procesos y las actividades de relacionadas con la organización y a los flujos de trabajo. Se centra el foco en optimizar todos los procesos y revisan constantemente los objetivos en el ámbito de calidad y rendimiento.

Dentro de los niveles de madurez identificados previamente, se hará énfasis en el **nivel 2** de madurez (gestionado); en el que hay áreas de proceso[8] que se ajustan perfectamente a algunas prácticas de la agilidad y que incluso dan ideas de mejora sobre las mismas.

En la figura 10 podemos observar todas las áreas de proceso de todo el modelo CMMI asociado a su categoría y nivel de madurez:

[8] Grupo de prácticas relacionadas en un área que, cuando se implementan de forma conjunta, satisface un conjunto de metas consideradas importantes para realizar mejoras en esa área.

| *Área de Proceso* | *Categoría* | *Nivel de Madurez* |
|---|---|---|
| Análisis Causal y Resolución (CAR) | Soporte | 5 |
| Gestión de Configuración (CM) | Soporte | 2 |
| Análisis de Decisiones y Resolución (DAR) | Soporte | 3 |
| Gestión Integrada del Proyecto (IPM) | Gestión de proyectos | 3 |
| Medición y Análisis (MA) | Soporte | 2 |
| Definición de Procesos de la Organización (OPD) | Gestión de procesos | 3 |
| Enfoque en Procesos de la Organización (OPF) | Gestión de procesos | 3 |
| Gestión del Rendimiento de la Organización (OPM) | Gestión de procesos | 5 |
| Rendimiento de Procesos de la Organización (OPP) | Gestión de procesos | 4 |
| Formación en la Organización (OT) | Gestión de procesos | 3 |
| Integración del Producto (PI) | Ingeniería | 3 |
| Monitorización y Control del Proyecto (PMC) | Gestión de proyectos | 2 |
| Planificación del Proyecto (PP) | Gestión de proyectos | 2 |
| Aseguramiento de la Calidad del Proceso y del Producto (PPQA) | Soporte | 2 |
| Gestión Cuantitativa del Proyecto (QPM) | Gestión de proyectos | 4 |
| Desarrollo de Requisitos (RD) | Ingeniería | 3 |
| Gestión de Requisitos (REQM) | Gestión de proyectos | 2 |
| Gestión de Riesgos (RSKM) | Gestión de proyectos | 3 |
| Gestión de Acuerdos con Proveedores (SAM) | Gestión de proyectos | 2 |
| Solución Técnica (TS) | Ingeniería | 3 |
| Validación (VAL) | Ingeniería | 3 |
| Verificación (VER) | Ingeniería | 3 |

*Figura 10. Áreas de proceso, categorías y niveles de madurez. [35]*

No se bajará al nivel de detalle de cada una de las áreas de proceso de cada nivel de madurez, pero se centrará la atención a las siguientes áreas del nivel de **madurez 2** que serán necesarias para las recomendaciones posteriores del documento:

- **Aseguramiento de la calidad del proyecto y del producto (PPQA):**

En esta área de proceso se establecen políticas con las expectativas y los medios de evaluación que usara la organización para asegurar que se cumplan los requerimientos tanto de procesos como de productos, junto con los métodos para solventar los errores o las inconformidades que se puedan generar.

Las políticas creadas en esta área de negocio van relacionadas con asegurar la calidad del producto y el proceso de manera independiente a la gestión del proyecto con la finalidad de hacer lo más objetivo posible la identificación de posibles errores.

- **Gestión de requisitos (REQM)**

Las políticas aquí definidas establecen las expectativas de la organización para gestionar los requisitos del proyecto e identificar las inconsistencias entre los requisitos y la planificación. Además de identificar la metodología que se va a usar para promover la trazabilidad de los requisitos en todo el ciclo de vida del proyecto, desde su identificación hasta su desarrollo he incorporación.

- **Planificación del proyecto (PP)**

Establece las expectativas de la organización para estimar y gestionar la planificación de los proyectos, además es aquí donde se definen los pactos internos y externos para desarrollar el plan de gestión del proyecto, así como los roles que, incorporados al mismo, su implicación y los compromisos de cada uno.

## 4. Pruebas en el software

Antes de iniciar me gustaría referenciar la siguiente frase del libro "***How Google Tests Software, s. f.***"

"*Quality is not equal to test. Quality is achieved by putting development and testing into a blender and mixing them until one is indistinguishable from the other." [8]*

El riesgo es algo inherente a la vida, y esto no exceptúa el desarrollo del software, además dependiendo del software que se realice el coste de que un riesgo se materialice es extremo e inasumible, háblese del caso de un software de control de vuelo o de una central nuclear o de transacciones bancarias por nombrar algunos. Existen muchas maneras de minimizar el riesgo en el software ya sea con mejores procedimientos de desarrollo, verificación de código, sistemas de revisión de bugs y sobre todo realizar pruebas.

Realizar pruebas ha sido parte de todas las industrias desde el principio de las eras, se han probado barcos desde que el mundo es mundo y se han probado coches desde el inicio de la era industrial, aviones, cohetes, materiales de construcción y un infinito etc, existe una regla de oro en la realización de pruebas que es la siguiente:

"*Cuanto más grandes son los riesgos mejores deben ser las pruebas"*

Y si lo pensamos detenidamente, la calidad que existe en muchos de los productos que usamos, desde la comida que consumimos, hasta el material de nuestra casa lo relacionamos a que alguien o algo (ya sea una persona, una corporación o una máquina) ya lo ha probado lo suficiente como para garantizar que sea bueno, de calidad o apto para el consumo, uso o implantación.

**Pruebas tempranas ahorran dinero**

Los modelos de desarrollo de software siempre deben dedicar tiempo y recursos para la realización de diversas pruebas sobre los productos y sus funcionalidades. Desafortunadamente dependiendo de cuando se disponga de un producto suficientemente estable y completo para realizar pruebas, se corre el riesgo de postergar para el final la etapa de pruebas. Lo que puede desencadenar en presión a los responsables de las pruebas para realizarlas lo antes posible, minimizando de esta manera el tiempo con el que se cuenta para resolver posibles incidencias y repetir ciclos de pruebas completos.

Es por lo anterior que se debe defender la premisa que las pruebas tempranas no solo ahorran tiempo a todos los implicados en el proyecto, sino que eliminan ambigüedad y permiten adelantar la recopilación de evidencias, sino que da certeza y tranquilidad a los gestores de cualquier proyecto sobre el producto que se está creando.

La siguiente gráfica es simplemente orientativa para realizar una visualización del incremento desproporcionado de identificar y solventar un error en las diversas etapas del proyecto.

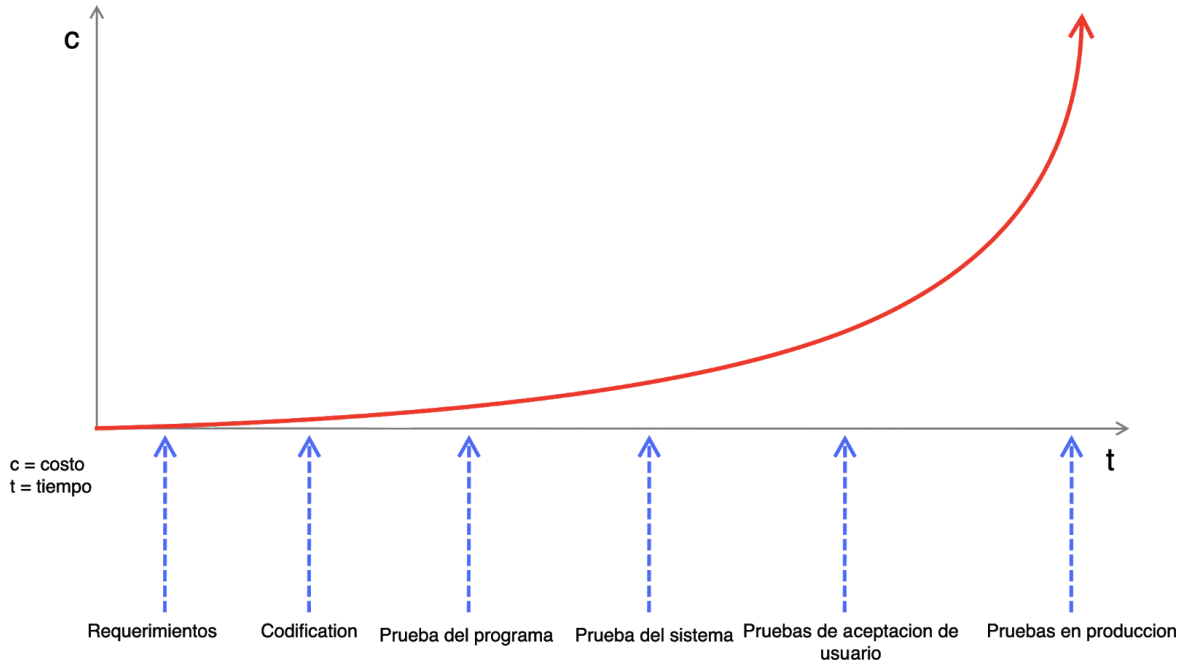


*Figura 11. Coste de identificación tardía de un error según la etapa del producto.*

**Niveles de pruebas**

Dependiendo del nivel del producto o el estado de los desarrollos, se barajan diversos niveles de prueba los cuales podemos ver a continuación:

- **Pruebas unitarias:** las cuales usualmente son realizadas por los mismos desarrolladores y cubren el nivel más bajo de los componentes o funcionalidades
- **Pruebas de integración:** las cuales se usan para verificar las relaciones entre componentes o entidades externas, las mismas a su vez prueban un amplio espectro de complejidades.
- **Pruebas de sistema:** permite verificar el sistema en su totalidad, junto con las interacciones que el sistema puede tener con entidades externas. Las mismas deben ser realizadas sobre funcionalidades completas.

- **Pruebas de aceptación:** le permite al cliente o a la persona delegada por él verificar el sistema en su totalidad, en el diagrama no se ve del mismo color que las pruebas realizadas por el gestor de pruebas o el desarrollador, ya que no se espera un nivel alto de verificación de todos los casos realizados y es más un grupo de pruebas básicas.

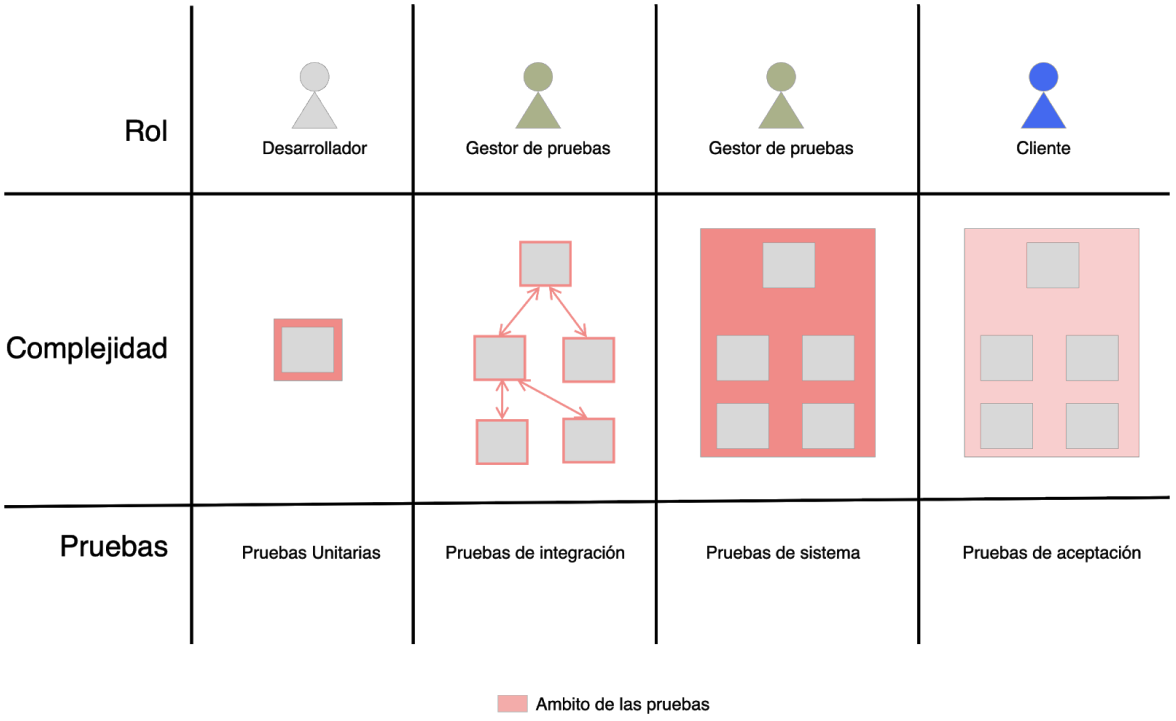


*Figura 12. Niveles de pruebas*

**Pruebas de integración, cuándo usarlas y qué aportan**

Después de que las pruebas unitarias hayan sido pasadas y/o escritas por el desarrollador en código para ser automatizadas, la siguiente etapa de las pruebas es la realización de pruebas integradas, en estas pruebas se verificaran los componentes que interaccionan ya sea con otros componentes ya desarrollados como con entidades externas siempre que sean necesarias para desarrollar funcionalidades en conjunto.

Las pruebas integradas mismas envuelven un gran número de pruebas a un nivel muy bajo de detalle y requieren conocimientos técnicos para conocer cómo se están integrando cada uno de ellos y los resultados esperados.

El objetivo de estas pruebas es exponer los defectos que pueden existir entre las interfases y las interacciones de los componentes integrados.

El objetivo de estas pruebas es el siguiente:

- Reducir riesgo
- Verificar si el comportamiento funcional y no funcional de las interacciones es el esperado.
- Generar confianza en la calidad de las integraciones.
- Identificar defectos ya sea en los componentes integrados como en las en las entidades externas.
- Prevenir que los defectos escalen o den una falsa percepción en su comportamiento.

## Casos Prácticos

A continuación, se desarrollarán un par de casos reales sobre los que se pueden realizar las recomendaciones sobre el framework Scrum para minimizar el riesgo y la incertidumbre para etapas posteriores, es decir, incertidumbre en el desarrollo como incertidumbre en los futuros mantenimientos.

### Contexto

Existe una tendencia en la actualidad por parte de muchas empresas a enfocar sus desarrollos software al desarrollo de microservicios e incluso migrar aplicaciones actuales a esta arquitectura por las diversas ventajas que esta arquitectura nos presta y que ya han sido exploradas anteriormente. Gracias a la implantación de estas nuevas arquitecturas también ha nacido la necesidad de crear aplicaciones frontales que consuman a los mencionados microservicios mediante aplicaciones (web's, móviles o tablets) de manera similar a la visualizada en el gráfico:

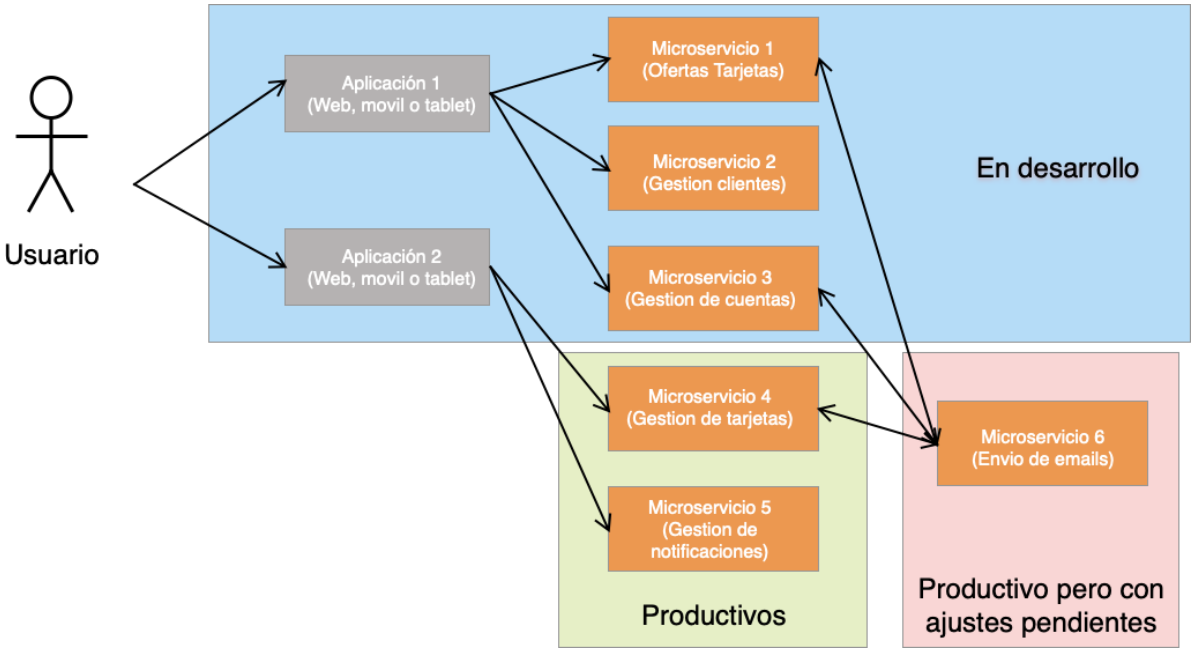


*Figura 13. Ejemplo de integración de microservicios con interfaces de usuario*

Como se puede inferir de la figura 13, gran parte de los beneficios de los microservicios requieren un alto nivel de gestión y control de estos previendo la

complejidad que pueden alcanzar. Lo anterior debido a que su arquitectura permite que cualquier servicio esté en fases diferentes del ciclo de vida al mismo tiempo.
Por ejemplo, el gráfico nos muestra cómo mientras se están desarrollando 2 interfases de usuario y 3 microservicios nuevos, hay 2 microservicios en etapas productivas seguramente siendo consumidos y 1 microservicio más en estado productivo, pero con ajustes pendientes para cumplir con los requerimientos del nuevo proyecto, obligando a realizar más pruebas para garantizar el no impacto de los consumidores.

Estos niveles de complejidad desencadenan muchas veces en errores, en los que los microservicios consumidos no cumplen completamente con todas las especificaciones, ya sea en cuanto a rendimiento, disponibilidad, como en funcionalidad.

Dependiendo del estado en que se encuentre el proyecto se inician procedimientos de búsqueda y corrección de los mencionados errores. Los procedimientos que deben finalizar en la corrección pueden ser “informales” en los que se simplemente se notifica una incidencia directamente al equipo de desarrollo y él mismo la réplica y la solventa, como con procedimientos “altamente formales”, que implican no solo reportar la incidencia sino, reuniones y planificaciones, recopilación de información e impacto, replica, corrección del error y planificación de la implantación para minimizar el impacto a los consumidores, etc.

Los casos siguientes, se expondrán teniendo en cuenta estos niveles de complejidad, aunque el **caso 1** solo se basa en el despliegue de componentes nuevos el **caso 2** algo más complejo puede implicar un desafío según el nivel de la organización y sus capacidades.

En este documento se plantean recomendaciones con base en experiencias reales que en conjunto evidenciaron la necesidad de mejorar los procedimientos de pruebas y documentos.

## Caso 1. ¿Es posible hacer proyectos mantenibles y escalables en el tiempo?

### Descripción del problema

Qué pasa cuando una empresa ha implantado el framework de Scrum para la realización de los productos software y desea que los productos desarrollados sean fácilmente mantenibles y escalables posterior a la entrega.

La verificación y el mantenimiento son costes inherentes a cualquier proyecto y deben ser planificados al inicio de cualquier proyecto esperando en que los mismos se mantenga en niveles asumibles durante el tiempo que se espere que el proyecto se mantenga productivo.

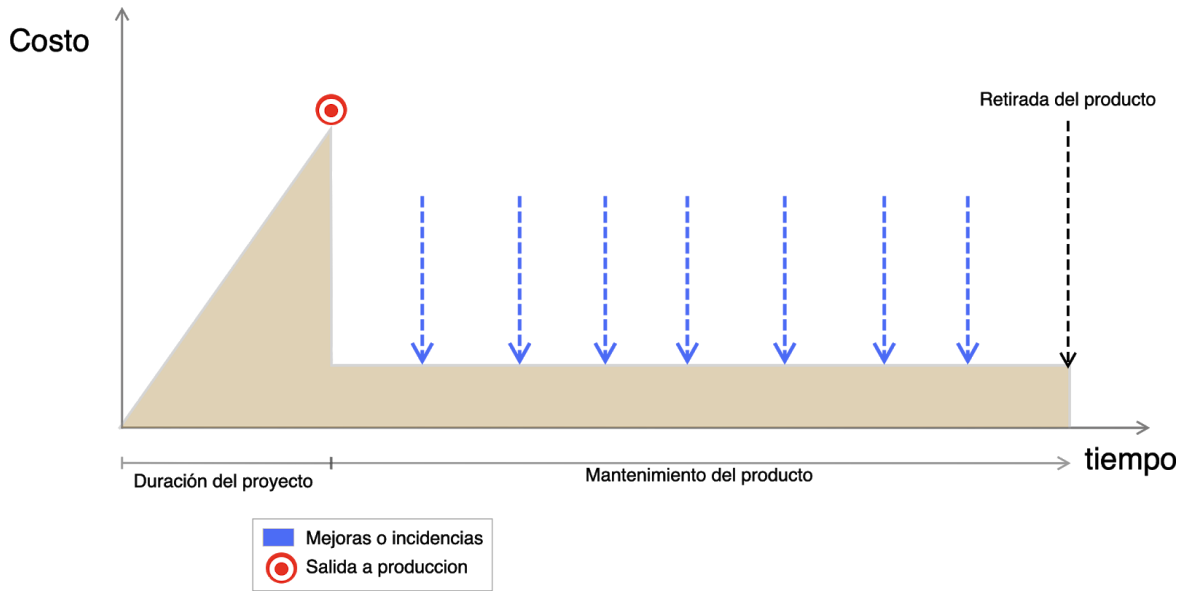


*Figura 14. Distribución ideal de costes de un proyecto*

Como se visualiza en la figura 14, el gasto en mantenimiento **ideal** es un valor estable posterior a la salida a producción o que se mantenga lo más posible dentro de parámetros cercanos al ideal.

Desafortunadamente la realidad en muchos proyectos es un diagrama similar al que se visualizará en la figura 15.

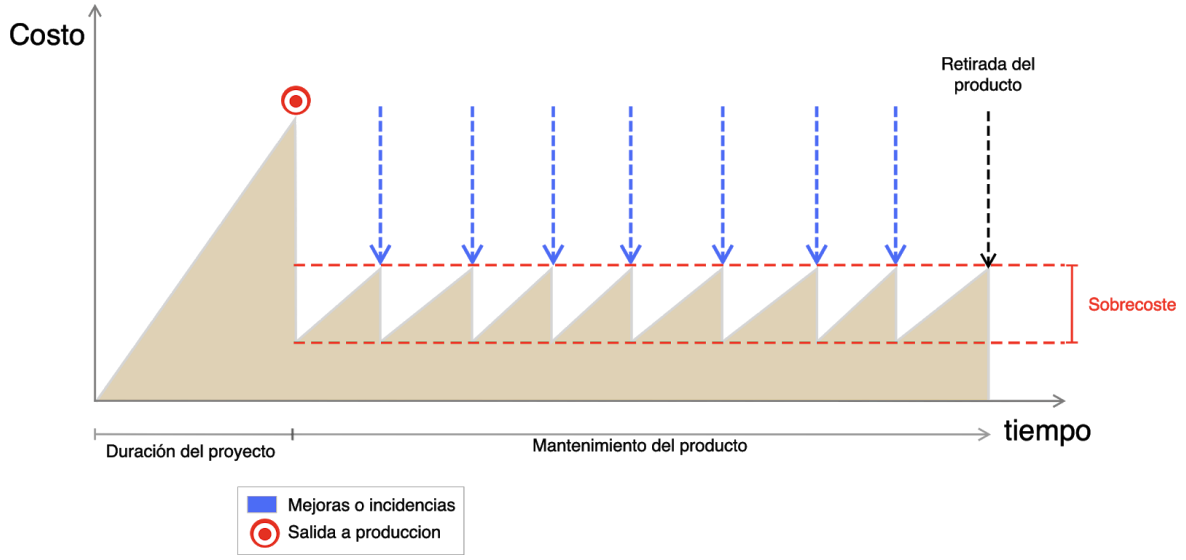


*Figura 15. Distribución real de costes de un proyecto cuando no se cuenta con documentación que técnica y de pruebas que le sustente.*

En donde los sobrecostes se materializan en el momento en que la salida a producción es una realidad. Los motivos de esos sobrecostes son muy variados, aunque solo por comentar algunos podemos destacar los siguientes:

- **Mala documentación técnica y de pruebas.** Que facilite la verificación e identificación del problema de manera efectiva.
- Tecnología de realización obsoleta o compleja.
- Pérdida de conocimiento (Adjudicada a la salida de las personas responsables de los desarrollos)
- Los cambios solicitados superan la potencialidad del producto y su arquitectura.

**Impacto del problema**

La falta de calidad siempre se traduce en costes; costes de tiempo, costes económicos, recursos, pérdida de imagen pública o de vidas si el sistema que los presenta errores es crítico. Los mencionados costes oscilan entre unidades, centenas o miles, dependiendo de en qué etapa del producto sea identificados los errores y los daños haya podido propiciar si fue en instancias productivas.

Cuando un producto está siendo desarrollado en Scrum, siempre se deben realizar pruebas de verificación de cada tarea de cada sprint, las mencionadas pruebas son parte del ciclo de vida Scrum. Actualmente por parte del framework es altamente recomendado realizar la mayor cantidad de pruebas automáticas sobre los componentes desarrollados.

Las pruebas automáticas tienen muchos beneficios, pero sobre todo su rapidez de réplica, ya que una vez codificadas es sorprendentemente fácil realizar la misma verificación de manera indefinida sobre la tarea en cualquier momento, con independencia del sprint en el que se encuentre el desarrollo o su avance. Sin embargo, se puede caer en el error de creer que estas pruebas suplen o son más importantes que las pruebas de funcionalidad y de aceptación del producto, pero sobre todo las pruebas que pretendo defender que son las **pruebas de integración** actualizadas y acorde a cada sprint.

**Recomendaciones**

En los equipos Scrum se cuenta con diversos roles desde el scrum Máster hasta el product Owner y el Development Team. Todos y cada uno de los roles de Scrum cumplen una función específica en el desarrollo del producto. Sin embargo, el carácter ambiguo del Development Team hace que en ocasiones sé pequé de alguna manera al no balancear totalmente los roles propios de cualquier desarrollo.

Como se comentó dentro del Development Team se requieren los desarrolladores que van a realizar el proyecto en la tecnología seleccionada, pero además existirán

personas que tengan otras habilidades que complementen lo requerido para el proyecto, ya sean responsables de bases de datos, administradores de servidores, desarrolladores backend, desarrolladores frontend, arquitectos de soluciones y responsables de pruebas.

Recordar que Scrum es un marco de trabajo, lo que implica que es la organización la que según su madurez, capacidad o políticas la que realiza las incorporaciones de los roles según sean requeridas en el proyecto, ya que como todos sabemos los roles del Development Team pueden no ser requeridos la totalidad de tiempo por el proyecto y pueden ser distribuidos en varios proyectos para maximizar su utilidad.

En el caso de no ser necesaria su implicación a tiempo completo en el proyecto se le puede permitir al miembro del equipo compaginar su tiempo con otros proyectos o tareas. Una distribución simplificada de tiempo en el caso del rol del responsable de pruebas puede ser la siguiente:


*Figura 16. Distribución de carga del responsable de pruebas en un sprint*

Se puede inferir que el nivel de carga rara vez se mantiene estable y dependiendo del proyecto y la etapa del ciclo de vida en el que se encuentre se puede requerir más o menos implicación. Debido a lo anterior se realizan las siguientes recomendaciones.

### Incorporar a un responsable de las pruebas desde el primer momento

Es un error no implicar al responsable de pruebas desde el momento inicial del proyecto, inclusive cuando se está realizando el análisis de los requisitos.

Esta idea la desarrollé del nivel 2 de CMMI, del área de proceso ***planificación del proyecto (PP)*** (Para mayor claridad referirse a la historia del arte), en la cual se seleccionan los roles que serán involucrados en los proyectos junto con las responsabilidades que cada uno tiene con el fin de asegurar todo lo necesario para desarrollar exitosamente el proyecto.

De ser el caso de no poseer toda la disponibilidad del tiempo para el proyecto, la implicación puede ser incremental sprint a sprint similar a lo que se ve en la figura 17.

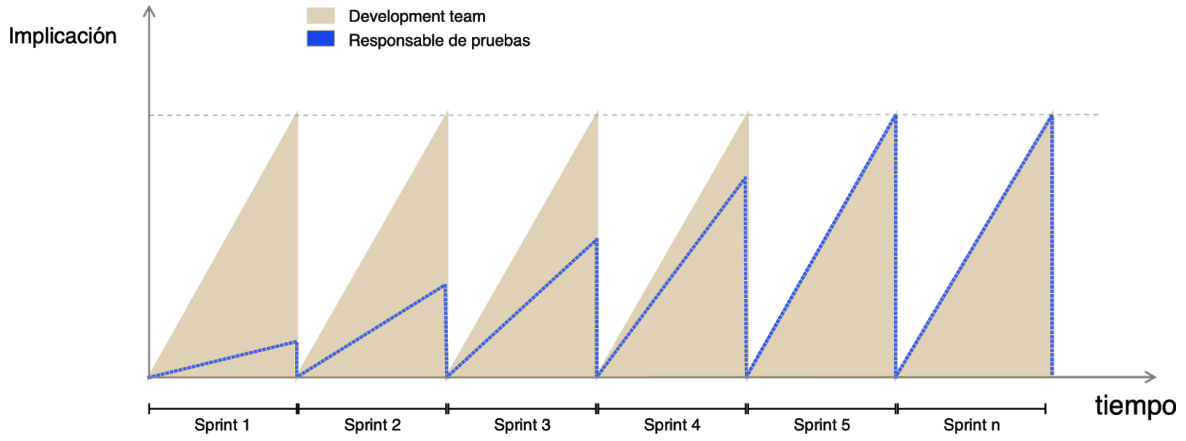


*Figura 17. Distribución de carga del responsable de pruebas por diversos sprints*

Como se puede apreciar, en los sprints iniciales la implicación puede ser menor, ya que las pruebas requeridas serán pocas al inicio mientras los desarrollos progresan e incrementan su tamaño y complejidad.

Pero el responsable de pruebas debe estar implicado en todo momento, sobre todo en las planificaciones que involucren adición, cambio o ajuste a las tareas del Product Backlog y Sprint Backlog. Además, particularmente en los análisis donde se revisa tarea a tarea y como se acometerán. Lo anterior para que el responsable de pruebas tenga una idea clara de que es lo que se debe probar y diseñe así los casos de prueba.

### Crear y actualizar el documento de pruebas de integración de manera incremental sprint a sprint

Como comentamos en la anterior recomendación, la implicación del responsable de las pruebas debe ser desde el inicio del proyecto o como mínimo desde la fase de análisis y definición.

La recomendación es iniciar el desarrollo del documento de pruebas de integración en desde las primeras etapas, el responsable de pruebas puede inicialmente trabajar casos a alto nivel e ir refinando caso a caso en la medida en que las tareas se vayan identificando y desarrollando en la medida que van pasando los sprints.

El documento de pruebas de integración debería cumplir una función similar a una bitácora del proyecto, el mismo no solo se almacenará parte de la historia del proyecto, sino que se debe mantener lo más actualizada y alineada posible con los

requerimientos, sin importar que los mismos evolucionen, cambien o se eliminen, el **cómo y él qué** se deben de reflejar siempre en este documento.

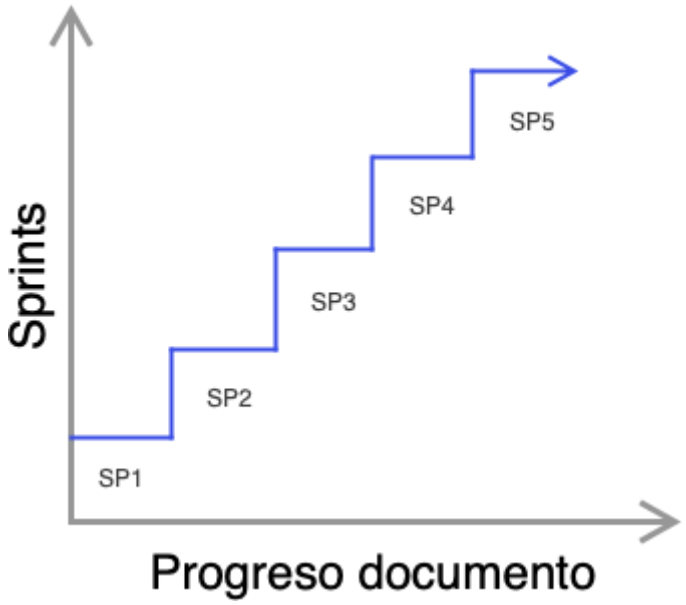


*Figura 18. Progreso documento de pruebas Sprint a Sprint (SP1...n)*

Como se ve en la figura 18 el documento incrementara sprint a sprint su tamaño, pero el mismo facilitará que se realicen pruebas rápidas sobre componentes ya verificados que a su vez revalidaran funcionalidades. El mismo también facilitara incorporar nuevos colaboradores en las pruebas simplemente siguiendo el documento.

El mencionado documento debe ser parte de la entrega del producto junto a toda la documentación técnica una vez finalice el proyecto y se entre en la etapa de mantenimiento. Disponer de una manera documentada y rápida de verificar una funcionalidad es vital a la hora de analizar una incidencia en las etapas de mantenimiento.

La idea de este documento la desarrollé del nivel 2 de CMMI, del área de proceso ***Gestión de requisitos (REQM)*****,** la cual nos insta a tener manera de realizar la

trazabilidad de los requisitos en todo momento desde su desarrollo, despliegue y solución como su retiro o eliminación del proyecto.

### Tener un documento de pruebas de integración por microservicio o por aplicaciones que le consumen

Como vimos en el contexto del problema los microservicios son componentes independientes que proveen funcionalidades variadas pero que pueden interaccionar entre ellos para complementar los casos de uso de otros proyectos.

Los mismos pueden ser usados por uno o muchos proyectos dependiendo de la funcionalidad que realicen Ej: un microservicio que envía emails de notificación es de imaginar que un microservicio con esas características puede ser usado de manera independiente por muchas aplicaciones web, aplicaciones móviles o incluso otros microservicios que requieran que se realice la funcionalidad que se provee.

Partiendo de la necesidad de integración y de que los microservicios deben ser lo más independiente posible es necesario crear documentos de pruebas de integración individuales para cada uno de ellos, en el ejemplo de la *figura 11* por ejemplo, deberíamos tener un documento por cada microservicio y por cada aplicación la cual garantizará la cobertura completa y real de todos los componentes que comprendieron el proyecto, debiendo realizar las siguientes acciones por cada documento en nuestro proyecto:

| | **Documento de pruebas de integración** |
|---|---|
| Aplicación 1 (Web, móvil o tablet) | **CREAR** |
| Aplicación 2 (Web, móvil o tablet) | **CREAR** |
| Microservicio 1 | **CREAR** |

| | |
|---|---|
| Microservicio 2 | **CREAR** |
| Microservicio 3 | **CREAR** |
| Microservicio 4 | No acción requerida |
| Microservicio 5 | No acción requerida |
| Microservicio 6 | **ACTUALIZAR** |

## Caso 2. Con un proyecto en producción y sin documentación de pruebas ¿Qué se hace?

### Descripción del problema

El problema para estos equipos de gestión de errores en etapas productivas nace cuando los microservicios, componentes o aplicaciones llevan ya tiempo productivos no tienen una fuente clara de conocimiento o el mismo se encuentra disperso en documentación escasa o en el peor de los casos inexistente.

Desafortunadamente dependiendo del tamaño y la madurez de la organización este es un problema que se ve con alguna regularidad, se cuenta erróneamente con que el conocimiento existe en la que la persona o personas creadoras y que las mismas siempre estarán dispuestas a solventar estos errores en cuanto aparecen o simplemente se fían en que nunca fallarán y que el conocimiento que se posee es suficiente para superar cualquier problema.

Pero independientemente de que el error se solvente, ¿cómo se verifica que se esté brindando exactamente el mismo servicio que se tenía disponible antes del error? ¿Cómo se da la garantía que la corrección de un error no desencadene en errores en otras aplicaciones o microservicios que lo consumen? ¿Quién debe garantizar eso y cómo puede realizar su trabajo de la mejor manera posible?

### Impacto del problema

El problema antes mencionado se materializa en los consumidores de los microservicios, ya sean aplicaciones (web o móviles), otros microservicios o entidades externas. A los cuales los fallos desencadenados les pueden acarrear diversos problemas, ya sean otros errores, fallos en cascada o inoperatividad en el caso de servicios ya productivos además de retrasos o re-trabajo en el caso de

desarrollos nuevos que necesiten consumir el microservicio que está dando problemas.

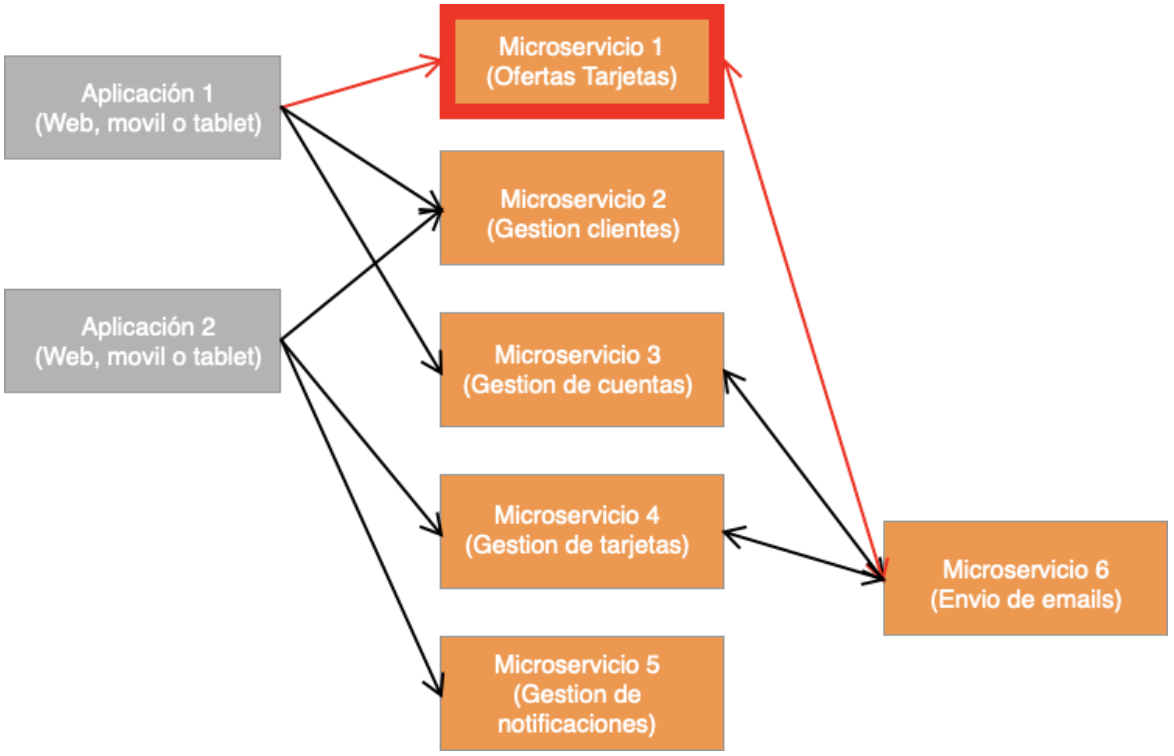


*Figura 19. Fallo en arquitectura de microservicios*

En la figura 19, se visualiza como un error en el microservicio 1 desencadena un efecto “bola de nieve” en los otros componentes que le consumen o interaccionan y en la interfaz de usuario que le consume directamente.

Los impactos de los errores son tan costosos como el nivel de criticidad del microservicio donde se detecte, si el sistema es crítico las consecuencias pueden ser incalculables Ej: transferencias electrónicas, pago de servicios públicos, etc. Pero si el microservicio provee funcionalidades menores o realiza tareas de bajo impacto sus errores pueden tener bajo impacto o recuperables de ser necesario.

**Recomendaciones**

La recomendación es la misma que en el caso anterior, pero con la complejidad adicional de que no puede ser desarrollado de manera incremental, sino que debe ser realizado con las herramientas de las que se disponga.  De igual manera, es necesario crear los documentos de pruebas integradas o solicitarlo de ser posible a los responsables del proyecto. Y en caso de no tenerlos crearlos a la mayor brevedad antes de que se manifiesten errores.

Partiendo de que no se cuenta con documentación distinta a la solicitada expresamente en el proyecto, se descarta contar con documentación extensiva y estructurada de pruebas las acciones recomendadas serían las siguientes.

### Asignación de equipo de calidad para la verificación casos críticos

La recomendación para el caso de no contar con documentación para verificar el correcto funcionamiento de un componente **crearla**, el cómo y el cual lo veremos a continuación, pero lo que sí se debe es de contar con un equipo o persona que cree este documento con todas las funcionalidades críticas incluidas por cada microservicio.

### Analizar y verificar toda la documentación con la que se cuente

Sé que la palabra clave de este documento es la imperiosa necesidad de tener una documentación estructurada de pruebas y que precisamente es la falta de esta misma documentación, el factor clave por el que nos encontramos en una situación así.

Pero hay que tener en cuenta que un proyecto Scrum genera cantidades ingentes de documentación, por nombrar algunas:

- Tareas del Sprint backlog (que pueden o no estar almacenadas en herramientas similares al JIRA o Trello por nombrar algunas),
- Pruebas unitarias de desarrollo (aunque estén codificadas)
- Pruebas automáticas codificadas en las diversas herramientas con las que se cuenta para este fin.
- El código fuente puede ser usado como documentación.
- Documento funcional de requisitos.
- Documentos de arquitectura.

El problema es precisamente que existe mucha información para las pruebas y que la misma debe estar condensada lo más posible en un documento con esta finalidad. La recomendación que se realiza en unas primeras instancias es recopilar toda la documentación con la que se cuenta, para con esa información iniciar la creación y/o actualización en el caso de que se cuente con ella de todos los casos de prueba como mínimo los críticos del producto.

### Crear el documento con los casos de prueba críticos

La documentación que se puede crear actualmente asociada a las pruebas es bastante extensa desde documentos de escenarios, test unitarios, pruebas end to end, pruebas de integración, etc. A continuación, se visualizan algunas de las técnicas de pruebas que se pueden usar y documentadas por la ISO y la IEEE.

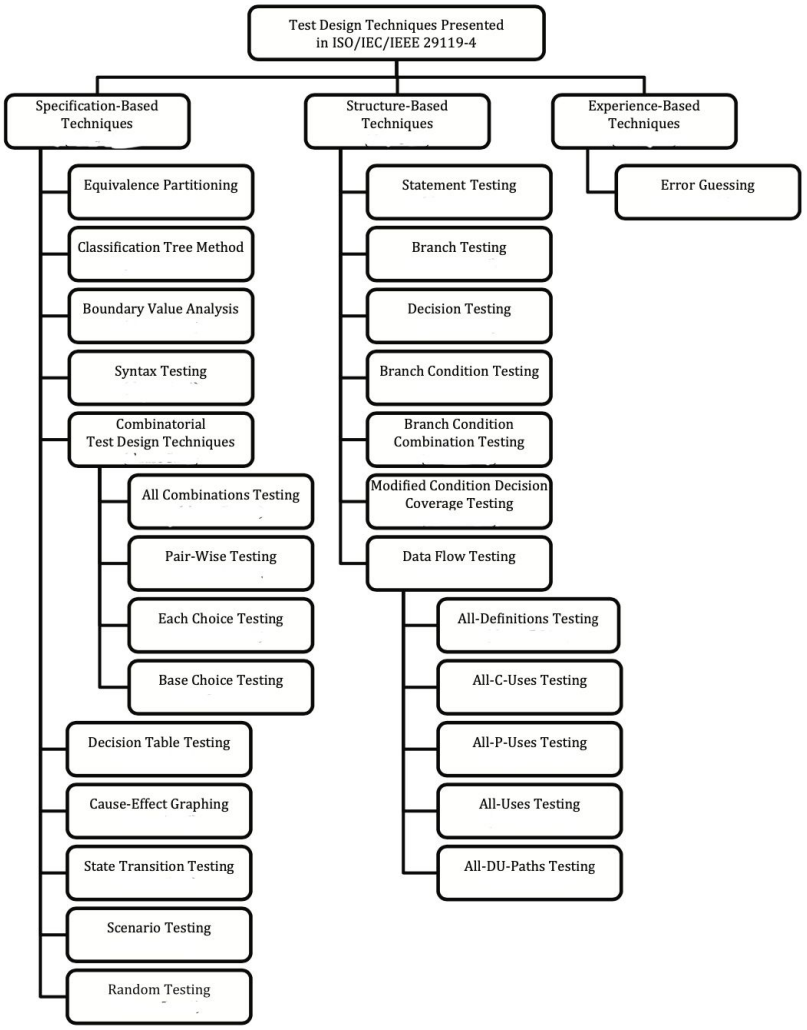


*Figura 20. Técnicas de testing documentadas en ISO/IEC/IEEE 29119-4 [15]*

La figura 20 muestra diversas técnicas de prueba en muchos niveles, algunos sumamente técnicos y otros a más alto nivel, en este documento no pretendo detallar las múltiples técnicas, pero sí destacar que dependiendo del nivel de criticidad del proyecto pueden ser requeridas más de una de las antes mencionadas y dependiendo de la organización algunas de estas técnicas son obligatorias. Como por ejemplo las pruebas que se realizan a la seguridad de las aplicaciones (Hacking Etico[9]) donde pretende vulnerar la aplicación para identificar posibles fallos.

[9] El hacking ético es una forma de referirse al acto de buscar las vulnerabilidades que tiene una aplicación, la búsqueda se realiza normalmente bajo demanda por personas calificadas para esta finalidad. Una vez identificadas las vulnerabilidades las mismas catalogadas y reportadas a los responsables de la aplicación para que sean solventadas.

Para el caso que nos compete de la figura 19 buscaríamos realizar el documento de pruebas de integración del microservicio 1, con toda la información recopilada buscando obtener la data para realizar las primeras versiones del documento de pruebas de integración y con esta información realizar pruebas en las que se verifique la relación entre cada componente.

Esta idea fue desarrollada del nivel 2 de CMMI, del área de proceso **Aseguramiento de la calidad del proyecto y del producto (PPQA)**, en la cual recomiendan políticas con los medios de evaluación de que se cumplen los requisitos que garantizan la operación correcta de todas las funcionalidades del producto.

Descomponiendo el diagrama anterior lo podríamos ver de la siguiente manera en casos:

**Caso 1:**

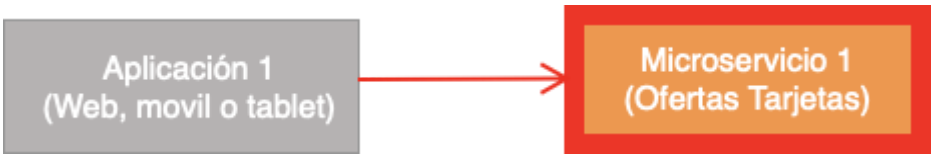


*Figura 21. Integración entre aplicación 1 y microservicio 1*

Caso de prueba # 1 interacción # 1

| | |
|---|---|
| **Sistema:** Alta de ofertas de tarjetas | **Caso de prueba:** Tiene el cliente una oferta |
| **Diseñado por:** Diego Esquivel | **Fecha de diseño:** |
| **Ejecutado por:** Diego Esquivel | **Fecha de ejecución:** |
| **Versión de probada:** v.0.1 | **Prueba sobre microservicio (S/N)** **N**<br>**Prueba sobre aplicación (S/N)** **S** |

**Descripción corta:**

Se verificará como se visualiza por parte del cliente una oferta de tarjeta.

**Precondiciones:**

El cliente debe estar autenticado.

El cliente debe tener una oferta asignada en el sistema.

| Paso | Acción | Respuesta | OK / KO | Referencias | Comentarios |
|---|---|---|---|---|---|
| 1 | Acceder con un cliente a la página de oferta | Se debe visualizar una oferta. | OK | • Caso de prueba # 3 interacción # 1 | |
| 2 | En la página de la oferta. | Se debe visualizar una oferta con el siguiente formato:<br>- Imagen de la tarjeta<br>- Beneficios<br>- Cuota<br>- Cupo | OK | • Caso de prueba # 2 interacción # 1<br>• | |
| 3 | Seleccionar la oferta | Se debe visualizar en la parte superior derecha la oferta seleccionada y habilitar el botón continuar. | OK | • Caso de prueba # 2 interacción # 1 | |
| 4 | | | | | |
| 5 | | | | | |

| **Post-Condición:**<br>Una vez finalizada estas interacciones se debe de continuar con el caso 2 |
|---|

**Historial de cambios**

| Fecha de realización | Localización de detalles del cambio | Comentarios / Observaciones |
|---|---|---|
| | | |

**Historial de comentarios**

| Fecha de realización | Comentarios / Observaciones |
|---|---|
| | |

**Caso 2:**

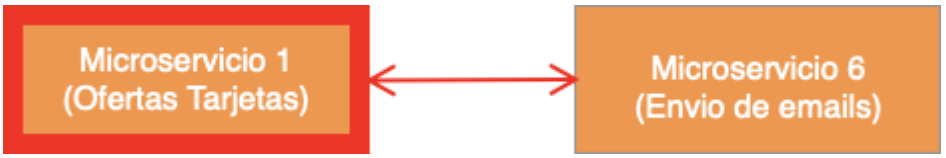


*Figura 22. Integración entre microservicio 1 y microservicio 6*

Caso de prueba # 2 interacción # 1

| | |
|---|---|
| **Sistema:** Alta de ofertas de tarjetas | **Caso de prueba:** Solicitar oferta vía email. |
| **Diseñado por:** Diego Esquivel | **Fecha de diseño:** |
| **Ejecutado por:** Diego Esquivel | **Fecha de ejecución:** |
| **Versión de probada:** v.0.1.1 | **Prueba sobre microservicio (S/N)** **S**<br>**Prueba sobre aplicación (S/N)** **N** |

**Descripción corta:**

Se invocará la funcionalidad de solicitar oferta directamente desde el microservicio 1 y se obviará el uso de la aplicación 1 para esta finalidad. Recordar que un microservicio puede ser invocado mediante un API o mediante ejecuciones directas y todo depende de la arquitectura.

**Precondiciones:**

El cliente debe estar autenticado.

El cliente debe tener una oferta asignada en el sistema.

| Paso | Acción | Respuesta | OK / KO | Referencias | Comentarios |
|---|---|---|---|---|---|
| 1 | Obtener la oferta | Se debe visualizar una oferta. | OK | ● Caso de prueba # 1 interacción # 1 | |
| 2 | Seleccionar oferta | Se debe de seleccionar una oferta de las obtenidas previamente. | OK | ● Caso de prueba # 2 interacción # 1 | Almacenar datos básicos de la oferta como el monto ofertado y el tipo de tarjeta. |
| 3 | Solicitar oferta | Se debe de solicitar la oferta mediante el servicio habilitado para este fin | OK | ● Caso de prueba # 3 interacción # 1 | |
| 4 | Revisar el email con la confirmación de la solicitud | Debe de llegar un email con la confirmación de la solicitud de la oferta | OK | ● Caso de prueba # 3 interacción # 1 | |

| | | | | | |
|---|---|---|---|---|---|
| | | previamente seleccionada | | | |
| 5 | | | | | |

**Post-Condición:**

Una vez finalizada estas interacciones se debe de continuar con el caso 3

**Historial de cambios**

| Fecha de realización | Localización de detalles del cambio | Comentarios / Observaciones |
|---|---|---|
| | | |

**Historial de comentarios**

| Fecha de realización | Comentarios / Observaciones |
|---|---|
| | |

**Caso 3:**


*Figura 23. Integración entre aplicación 1, microservicio 1 y microservicio 6*

Caso de prueba # 3 interacción # 1

| | |
|---|---|
| **Sistema:** Alta de ofertas de tarjetas | **Caso de prueba:** Tiene el cliente una oferta, la selecciona y la solicita. |
| **Diseñado por:** Diego Esquivel | **Fecha de diseño:** |
| **Ejecutado por:** Diego Esquivel | **Fecha de ejecución:** |
| **Versión de probada:** v.0.1.1 | **Prueba sobre microservicio (S/N)** **S**<br>**Prueba sobre aplicación (S/N)** **S** |

**Descripción corta:**

Desde la aplicación el cliente debe acceder a su cuenta, ver ofertas de tarjeta, seleccionar una de las ofertas y solicitarla. Posterior a eso le debe llegar un email confirmando la oferta solicitada.

**Precondiciones:**

El cliente debe estar autenticado.

El cliente debe tener una al menos una oferta asignada en el sistema.

| Paso | Acción | Respuesta | OK / KO | Referencias | Comentarios |
|---|---|---|---|---|---|
| 1 | Acceder con un cliente a la página de oferta | Se debe visualizar una oferta. | OK | ● Caso de prueba # 1 interacción # 1<br>● Caso de prueba # 2 interacción # 1 | |
| 2 | En la página de la oferta. | Se debe visualizar una oferta con el siguiente formato:<br>- Imagen de la tarjeta<br>- Beneficios<br>- Cuota<br>- Cupo | OK | ● Caso de prueba # 1 interacción # 1<br>● Caso de prueba # 2 interacción # 1 | |
| 3 | Seleccionar la oferta | Se debe visualizar en la parte superior derecha la oferta seleccionada y | OK | ● Caso de prueba # 1 interacción # 1<br>● Caso de prueba # 2 interacción # 1 | |

| | | | | | |
|---|---|---|---|---|---|
| | | habilitar el botón continuar. | | | |
| 4 | Solicitar oferta | Se debe de solicitar la oferta. | OK | ● Caso de prueba # 2 interacción # 1 | |
| 5 | Revisar el email con la confirmación de la solicitud | Debe de llegar un email con la confirmación de la solicitud de la oferta previamente seleccionada | OK | ● Caso de prueba # 2 interacción # 1 | |

| **Post-Condición:** |
|---|
| Una vez finalizada estas interacciones se debe de continuar con el caso 2 |

**Historial de cambios**

| Fecha de realización | Localización de detalles del cambio | Comentarios / Observaciones |
|---|---|---|
| | | |

**Historial de comentarios**

| Fecha de realización | Comentarios / Observaciones |
|---|---|
| | |

*Es importante tener en cuenta que la data usada para la realización de estos cuadros es data completamente inventada y no hace referencia a ningún proyecto real.*

Como recomendaciones de cada caso del documento realizaría las siguientes:

1. Usar un control de versiones del documento.
2. Solo se cambia la "**Versión probada"** del caso cuando se realizan las pruebas del mencionado caso y se debe corresponder con la versión que ha verificado entregada por el equipo de desarrollo.
3. Siempre se deberían probar todos los pasos del caso de pruebas y no de manera individual.
4. Se pueden tener tantos casos de prueba como sea necesario para verificar las interacciones.
5. Mantener actualizado el historial de cambios.

## Conclusiones y Recomendaciones Generales

La agilidad es parte del día a día en el desarrollo de productos software. En la actualidad y como se mostró en uno de los gráficos, el framework ágil más usado es Scrum y es este el framework con el que se trabajó en este TFM, aunque es importante destacar que todos los frameworks y metodologías ágiles nacieron con la misma premisa, ser ágiles, reaccionar rápido, con eficacia al cambio y aportar valor de calidad en el menor tiempo posible.

Los beneficios de la agilidad son innegables y cada persona que haya podido trabajar con alguno de los frameworks o metodologías de esta manera puede dar opiniones o posibles mejoras. Sin embargo, está demostrado que los productos software independientemente de su tecnología y el área donde sea usado ej. Banca, defensa, turismo, gobierno, etc o los equipos que lo desarrollan tienen un ciclo de vida hasta su retirada.

Es por lo anterior que cualquier herramienta y/o documentación que facilite la verificación de la funcionalidad durante el desarrollo del producto y el posterior mantenimiento en fases productivas siempre será de gran utilidad para los responsables del mismo.

Desafortunadamente una de las premisas del manifiesto ágil es:

"**Software funcionando** sobre documentación extensiva" [25]

En contraprestación tenemos los modelos llamados "tradicionales" de desarrollo de software y una de sus metodologías más extendidas es CMMI, el cual nos presenta su propio conjunto de recomendaciones para el desarrollo de proyectos. Para el mismo la documentación es obligatoria, pues es una de las bases de la calidad del producto, así como de la trazabilidad de los desarrollos.

CMMI en su nivel de madurez 2 nos presenta 3 áreas de proceso específicamente creadas para asegurar la calidad, la trazabilidad y las personas adecuadas para realizar las tareas necesarias y de esta manera garantizar la calidad de los productos desarrollados.

Además, hace un especial énfasis en que se esté documentando de manera incremental y que los documentos evolucionen en la medida en que el proyecto crezca, para de esta manera garantizar la verificación, el seguimiento el esperado éxito en los desarrollos.

Claro que muchas personas que trabajan modelos ágiles de desarrollo dirán que en los modelos tradicionales es más fácil de documentar debido a que los requerimientos cambian poco. Pero lejos de discutir esa premisa se hace la observación que todos los modelos deben permitir cambios en diferentes momentos del ciclo de vida del desarrollo y que la diferencia entre un modelo ágil y un modelo tradicional puede estar en la velocidad con la que se acometen los mencionados cambios o el nivel de trabajo que pueden requerir o los costes implicados en el cambio, pero que los mismos deben ser acometidos si el o los clientes lo solicitan independiente del modelo con el que se desarrolle.

Sobre el modelo CMMI es clara la necesidad de documentar las pruebas y los casos como parte de la verificación del producto y como esa documentación incremental puede aportar un grado de verificación de los requerimientos a los desarrollos ágiles

y particularmente al framework scrum que es el contexto en el que se desarrolló este TFM.

No pretendo discutir la necesidad de tener o no documentación y qué documento es o no necesario en un proyecto software porque siempre existen opiniones ya sea de coste, tiempo dedicación, entre otros. El dilema es que la premisa de la agilidad de documentar lo mínimo indispensable pudo haber legitimado a muchos equipos para que no desarrollen prácticamente ninguna documentación o que la misma no se preserve y se disgregue de las fases de desarrollo a las de producción, en donde también es necesaria.

Todos estamos de acuerdo en que una documentación excesiva puede no ser necesaria y que consume mucho tiempo, solo por poner un ejemplo en un proyecto normal pequeño de aproximadamente 3 meses se generan cantidades ingentes de información y además toda esa información puede sufrir cambios y/o evoluciones en el transcurso de cada interacción. Y qué se hace entonces, si parte de la agilidad es que los objetivos pueden cambiar y que los requisitos mutan, evolucionan o se eliminan en cualquier momento del desarrollo, lo que hace que la trazabilidad de un requisito sea un trabajo altamente complejo.

Mis recomendaciones van orientadas al framework scrum que es donde tengo más experiencia pero puede que personas con más conocimiento en otras metodologías o frameworks puedan adaptarlo a ellas y está enfocado en que el responsable o los responsables de las pruebas se deben incorporar al principio del proyecto en la medida de la disponibilidad que posea, ya sea para aportar en el análisis como para estar informado de las tareas que se acometen por parte del equipo de desarrollo además de las soluciones que plantean para las tareas.

El responsable de las pruebas debería asistir a todas las reuniones que hacen parte de scrum, no solo para hacer parte del equipo sino para que pueda tener tareas paralelas complementarias a las acometidas en las que dedique tiempo al análisis de cada requerimiento y la definición del cómo se acometerá cada prueba.

Todas las tareas anteriormente mencionadas deberían ayudar a complementar un documento de pruebas, en el caso de este TFM he recomendado encarecidamente el documento de pruebas de integración, pero el mismo puede ser de pruebas de sistema, de aceptación o meramente funcionales pero lo importante es que sea un documento actualizado, que tenga la mayor cantidad de casos de uso y documentación de como verificarlos.

Este documento será parte de la entrega del producto y no solo ayudará a la verificación en las diversas fases del ciclo de vida, sino que será entregado y actualizado de ser necesario en etapas posteriores a la entrega del producto o en la mencionada fase de mantenimiento, permitiendo que se ahorren costes al momento de realizar verificaciones de incidencias o mejoras en las pruebas de no impacto.

## Glosario de Términos

- **Áreas de procesos (CMMI):** Un grupo de prácticas relacionadas en un área que, cuando se aplique colectivamente, satisface un conjunto de objetivos considerados importantes para la fabricación de mejora en esa área. https://es.qwe.wiki/wiki/Process_area_(CMMI)
- **API:** Es la abreviatura de Application Programming Interfaces, que en español significa interfaz de programación de aplicaciones. Es una especificación formal que establece cómo un módulo de un software se comunica o interactúa con otro para cumplir una o muchas funciones.
- **Bugs:** Se define como un error o falla en el software.
- **Responsables de pruebas:** Es el rol responsable de realizar el análisis, diseño y ejecución de las pruebas.
- **Arquitectos de soluciones:** Es un profesional con la visión end-to-end de la solución. Es el encargado de diseñar una solución para los requerimientos del cliente, mapeando los requerimientos funcionales hacia tecnologías.
- **JIRA:** Software diseñado originalmente como un gestor de incidencias y errores. Sin embargo, se ha convertido en una herramienta de gestión de trabajo para todo tipo de casos de uso, desde la gestión de requisitos y casos de prueba hasta el desarrollo de software ágil. https://www.atlassian.com/es/software/jira/guides/use-cases/what-is-jira-used-for
- **Trello:** Es una solución fácil, gratuita, flexible y visual para gestionar proyectos a todo nivel. https://trello.com/about
- **Pruebas de escenarios:** Un escenario de test se define como cualquier funcionalidad que se pueda probar. También se denomina Condición de prueba o Posibilidad de prueba. El gestor de pruebas debe ponerse en el lugar del usuario final y descubrir los escenarios del mundo real y los casos de uso de la Aplicación bajo prueba. Las pruebas de escenarios utilizan escenarios para las pruebas. Los buscan ayudar a de una forma simple

probar los sistemas más complicados. https://www.guru99.com/test-scenario.html

- **Test unitarios:** Es un tipo de prueba de software en el que se prueban unidades o componentes individuales de un software. El propósito es validar que los componentes funcionen individualmente como se espera. Las pruebas unitarias se realizan durante el desarrollo (fase de codificación) de una aplicación por parte de los desarrolladores. Las pruebas unitarias aíslan una sección de código y verifican su exactitud. Una unidad puede ser una función, método, procedimiento, módulo u objeto individual. https://www.guru99.com/unit-testing-guide.html
- **Pruebas end to end:** Permiten identificar fallas en los frontales de las aplicaciones. Este nivel de pruebas es superior a las pruebas de integración porque requiere la verificación de funcionalidades completas independientemente de las relaciones que puedan existir en capas inferiores.
- **Pruebas de integración:** Son las pruebas usadas para verificar las relaciones entre componentes o entidades externas, las mismas a su vez prueban un amplio espectro de complejidades.
- **Hacking Etico:** Es una forma de referirse al acto de buscar las vulnerabilidades que tiene una aplicación, la búsqueda se realiza normalmente bajo demanda por personas calificadas para esta finalidad. Una vez identificadas las vulnerabilidades las mismas catalogadas y reportadas a los responsables de la aplicación para que sean solventadas.
- **Microservicios:** Son un tipo de arquitectura que sirve para diseñar aplicaciones. Lo que distingue a la arquitectura de microservicios de los enfoques tradicionales y monolíticos es la forma en que desglosa una aplicación en sus funciones principales. Cada función se denomina servicio y se puede diseñar e implementar de forma independiente. Esto permite que funcionen separados (y también, fallen por separado) sin afectar a los demás.
- **Interfaz de usuario:** es el medio con que un usuario puede comunicarse con una máquina, equipo, computadora o dispositivo.
- **Scrum:** es un marco de trabajo iterativo e incremental para el desarrollo de proyectos, productos y aplicaciones. El objetivo del mismo es estructurar el

desarrollo en ciclos de trabajo llamados sprints, los cuales son iteraciones de trabajo de entre 1 y 4 semanas, y los mismos se van sucediendo uno detrás de otro. Los sprints son de duración fija y terminan en una fecha específica, aunque no se hayan terminado las tareas que lo componen y nunca se alargan.

- **CMMI:** (Modelo de Madurez de Capacidades de Integración) permite medir o establecer qué tan madura es una organización o equipo de trabajo con base en cómo la misma implementa las recomendaciones del modelo, presenta una guía de características efectivas integrables en los procesos de las organizaciones proporcionando una guía de creación y mejora de los procesos internos.
- **EFQM:** Modelo Europeo de Gestión de Calidad (o Modelo Europeo de Excelencia) fue elaborado por la Fundación Europea para la Gestión de Calidad y está orientado al análisis de procesos con el fin de mejorarles y adaptarles mediante la implementación de prácticas que faciliten la autoevaluación y la mejora continua tanto en entornos empresariales privados como públicos. El modelo permite la certificación de las diversas entidades y es una certificación muy valorada en las entidades públicas de la unión europea.

## Bibliografía


1. Agile by the numbers. (s. f.). Deloitte Insights. Recuperado 20 de agosto de 2020, de https://www2.deloitte.com/us/en/insights/industry/public-sector/agile-in-government-by-the-numbers.html

2. BACómetro 2018. (2018, 25 abril). Business Agility Corporation. http://businessagilitycorp.com/bacometro/

3. De aplicaciones monolíticas a microservicios | Aplyca. (s. f.). Aplyca. Recuperado 28 de julio de 2020, de https://www.aplyca.com/es/blog/aplicaciones-monoliticas-o-microservicios

4. Market Research Future. (s. f.). Microservices Architecture Market Research Report- Forecast to 2023 | MRFR. https://www.marketresearchfuture.com/reports/microservices-architecture-market-3149

5. Flow diagram. (s. f.). Scrum.org. Recuperado 15 de julio de 2020, de https://www.scrum.org/forum/scrum-forum/39613/flow-diagram

6. The various levels of Services in the 3 Scrum Roles. (s. f.). Scrum.org. Recuperado 1 de agosto de 2020, de https://www.scrum.org/resources/blog/various-levels-services-3-scrum-roles

7. CMMI Maturity Levels - Tutorialspoint. (s. f.). Tutorialspoint. Recuperado 15 de agosto de 2020, de https://www.tutorialspoint.com/cmmi/cmmi-maturity-levels.htm

8. ¿Qué son los microservicios? (s. f.). RedHat. Recuperado 15 de junio de 2020, de https://www.redhat.com/es/topics/microservices/what-are-microservices

9. Whittaker, J. A., Arbon, J., & Carollo, J. (2012). How Google tests software. Addison-Wesley.

10. Samaroo, A., Thompson, G., & Hambling, B. (2015, June). Software Testing: An ISTQB-BCS Certified Tester Foundation Guide 3rd ed. BCS.

11. Rahman, M. M. (2019). Department of Software Engineering, FSIT SWE-431 Project/Thesis Project Documentation Supervised by (Doctoral dissertation, Daffodil International University).

12. Tshabalala, M. M. (2019, November). Impact of Agile Misconceptions Towards Organization Technology Competitiveness. In *2019 International Multidisciplinary Information Technology and Engineering Conference (IMITEC)* (pp. 1-8). IEEE.

13. Fernández, E. (s. f.). Blog de Edgar Fernandez. Edgar Fernández. Recuperado 18 de mayo de 2020, de https://www.edgarfernandez.com/blog/

14. Villán, V. R. (2020, 7 abril). Qué son las metodologías ágiles y cuáles son sus ventajas empresariales. Thinking for Innovation. https://www.iebschool.com/blog/que-son-metodologias-agiles-agile-scrum/

15. Proceso y Roles de Scrum. (s. f.). Softeng. https://www.softeng.es/es-es/empresa/metodologias-de-trabajo/metodologia-scrum/proceso-roles-de-scrum.html

16. BACómetro 2018. (2018, 25 abril). Business Agility Corporation. http://businessagilitycorp.com/bacometro/

17. Parody, L. (2019, 2 diciembre). How to Manage Modern Software Projects: Waterfall vs. Agile. Medium. https://medium.com/@lizparody/waterfall-vs-agile-methodology-in-software-development-1e19ef168cf6

18. Miyashiro, M. A. S., Ferreira, M. G., & Sant'Anna, N. (2015, July). CMMI-DEV process areas modeled on a process for critical embedded systems development. In *2015 Science and Information Conference (SAI)* (pp. 870-878). IEEE.

19. P29119-4-FDIS, Apr 2015 - IEEE Approved Draft International Standard for Software and Systems Engineering--Software Testing--Part 4: Test Techniques - IEEE Standard. (2015, 8 diciembre). Ieeexplore. https://ieeexplore.ieee.org/document/7091844

20. Meneses, Y. N. G., Padilla, N. Y. L., Mora, J. J. H., & Barrera, M. G. M. (2014). Análisis del estado actual de certificaciones CMMI-DEV ver. 1.3 año 2013 y 2014, a nivel mundial y en México. *Research in Computing Science*, *79*, 121-134.

21. Trigás Gallego, M. (2012). Metodologia scrum. Recuperado 7 de mayo de 2020, http://openaccess.uoc.edu/webapps/o2/bitstream/10609/17885/1/mtrigasTFC0612memoria.pdf

22. Kniberg, H. (2007). Scrum y XP desde las trincheras. *Estados Unidos: C4Media*.

23. Acevedo, C. A. J., y Jorge, J. P. G., & Patiño, I. R. (2017, October). Methodology to transform a monolithic software into a microservice

architecture. In *2017 6th International Conference on Software Process Improvement (CIMPS)* (pp. 1-6). IEEE.

24. Baldassarre, M. T., Caivano, D., Pino, F. J., Piattini, M., & Visaggio, G. (2012). Harmonization of ISO/IEC 9001: 2000 and CMMI-DEV: from a theoretical comparison to a real case application. *Software Quality Journal*, *20*(2), 309-335.

25. Manifiesto por el Desarrollo Ágil de Software. (s. f.). agilemanifesto. Recuperado 10 de mayo de 2020, de https://agilemanifesto.org/iso/es/manifesto.html

26. R. Nandakumar, A. K. Lal, and R. M. Parmar. 2014. State of the art in software quality assurance. SIGSOFT Softw. Eng. Notes 39, 3 (June 2014), 1–6. DOI: https://doi.org/10.1145/2597716.2597724

27. Ahmed, A., Ahmad, S., Ehsan, N., Mirza, E., & Sarwar, S. Z. (2010, June). Agile software development: Impact on productivity and quality. In 2010 IEEE International Conference on Management of Innovation & Technology (pp. 287-291). IEEE.

28. Savchenko, D. I., Radchenko, G. I., & Taipale, O. (2015, May). Microservices validation: Mjolnirr platform case study. In 2015 38th International convention on information and communication technology, electronics and microelectronics (MIPRO) (pp. 235-240). IEEE.

29. Perera, P., Silva, R., & Perera, I. (2017, September). Improve software quality through practicing DevOps. In 2017 Seventeenth International Conference on Advances in ICT for Emerging Regions (ICTer) (pp. 1-6). IEEE.

30. Figuera Perez, A. (2019). *Arquitectura de microserveis-Service Mesh* (Bachelor's thesis, Universitat Politècnica de Catalunya).

31. Silva, F. S., Soares, F. S. F., Peres, A. L., de Azevedo, I. M., Vasconcelos, A. P. L., Kamei, F. K., & de Lemos Meira, S. R. (2015). Using CMMI together with agile software development: A systematic review. *Information and Software Technology*, *58*, 20-43.

32. Salinas, C. T., Escalona, M. J., & Mejías, M. (2012, December). A scrum-based approach to CMMI maturity level 2 in web development environments. In Proceedings of the 14th International Conference on Information Integration and Web-based Applications & Services (pp. 282-285).

33. Torrecilla-Salinas, C. J., Sedeño, J., Escalona, M. J., & Mejías, M. (2016). Agile, Web Engineering and Capability Maturity Model Integration: A systematic literature review. Information and Software Technology, 71, 92-107.

34. La Importancia de la Arquitectura de Soluciones. (s. f.). SG Buzz. Recuperado 1 de agosto de 2020, de https://sg.com.mx/revista/46/la-importancia-la-arquitectura-soluciones

35. Equipo del Producto, C. M. M. I. (2010). CMMI® para Desarrollo, Versión 1.3. Software Engineering Institute.

36. K, Y. S., Gaurav, Y., Ashutosh, D., & T, H. H. (2017). Software testing integration.

37. EFQM.es: Modelo de excelencia y calidad EFQM. (s. f.). Fundación Europea para la Gestión de la Calidad. Recuperado 20 de agosto de 2020, de http://www.efqm.es/

38. Colaboradores de Wikipedia. (2020, 6 junio). Interfaz de usuario. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Interfaz_de_usuario